Cisco Crosswork Situation Manager 7.0.x Implementor Guide

Powered by Moogsoft AIOps 7.0.1

Implementor Guide

Use this guide to learn how to install Cisco Crosswork Situation Manager v6.5. If you are installing another version, see Cisco Crosswork Situation Manager Releases for more information.

Refer to the following topics to help choose the right environment for your Cisco Crosswork Situation Manager deployment:

- The Cisco Crosswork Situation Manager 7.0.0 Supported Environments topic details supported operating systems and system requirements.
- The Sizing Recommendations will help you make sure you select hardware to support your data ingestion and user requirements.

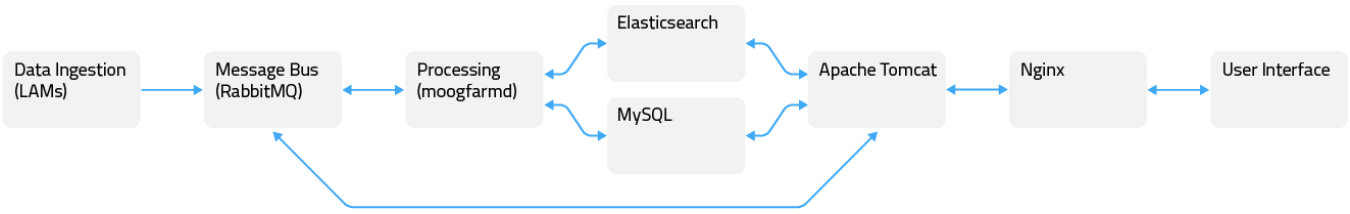
If you are upgrading Cisco Crosswork Situation Manager, see Upgrade - RPM Deployments.

Cisco Crosswork Situation Manager Packages

A Cisco Crosswork Situation Manager deployment comprises several packages. The different implementation procedures offer you flexible ways to deploy the packages, which consist of the following:

- Integrations (LAMs) that listen or poll for events, parse and encode them into discrete events, and then write the discrete events to the message bus.
- The message bus (RabbitMQ) that receives published messages from integrations. It publishes messages destined for data processing (moogfarmd) and the web application server.
- The system datastore (MySQL) that handles transactional data from other parts of the system: integrations (LAMs), data processing, and the web application server.
- The data processing component (moogfarmd), an application that consumes messages from the message bus. It processes event data in a series of servlet-like modules called Moolets. Moogfarmd reads and writes to the database and publishes messages to the bus.
- The web application server (Tomcat) that reads and writes to the bus and the database.
- A proxy (Ngnix) for the web application server and for integrations.
- The search engine (Elasticsearch) for the UI that indexes documents from the indexer moolet in the data processing series. It returns search results to Tomcat.

The diagram below shows the general data flow between components:



Deployment Options

You have the option to install all Cisco Crosswork Situation Manager packages on a single machine. However, the modular approach of the Cisco Crosswork Situation Manager distribution means fewer dependencies between individual packages. This means you have the flexibility to install different components to different machines. For example, you can install the web server components, Nginx and Tomcat, and the database component, MySQL, on different hosts.

- To quickly deploy Cisco Crosswork Situation Manager for evaluation or learning purposes, you can request trial
- access to a SaaS environment. For smaller deployments, you can run all the components in on a single machine.
    - If you have root access to the machine and want to use yum to install, see Single Host Installation - RPM.
    - If you do not have root access to the machine where you are installing and you want more control over where you install Cisco Crosswork Situation Manager, see Single Host Installation for Non-root.
- For larger deployments, you may install different components to different machines in order to distribute the workload. See Distributed Installation - RPM.

Cisco Crosswork Situation Manager 7.0.0 Supported Environments

The following operation systems, browsers and third-party software are either supported or are required in order to run Cisco Crosswork

Situation Manager. Any operating systems and browsers not listed in the sections below are not officially recommended or supported by

Cisco.

Operating Systems

You can run Cisco Crosswork Situation Manager on the following versions of Red Hat Enterprise Linux® (RHEL) and CentOS Linux:

OS	Versions
CentOS	v7
RHEL	v7

> No other Linux distributions are currently supported

Browsers

You can use the following browsers for the Cisco Crosswork Situation Manager UI:

	Version
Chrome	Latest
Firefox	Latest
Safari	Latest
Edge	Latest
IE	v11

Supported Third-Party Software

The latest default installation of Cisco Crosswork Situation Manager comes with the following third-party applications:

Application	Version
Apache Tomcat®	v8.0.48
Elasticsearch	v5.6.9
MySQL Community Server	v5.7.22
Nginx	v1.14.0 or above
RabbitMQ	v3.7.4

Other supported application packages include:

Application	Version
Erlang	v20.1.7
JRE	v1.8.0_171
Apache Tomcat® Native	v1.1.34 or above

Integration Support

The following table outlines the vendor supported integrations for the current version of Cisco Crosswork Situation Manager alongside the corresponding supported software versions:

Integration Version	Supported Software / Version
Apache Kafka Integration v1.6	Apache Kafka v0.9, 1.1
Ansible Tower Integration v1.6	Ansible Tower v3.0, 3.1
AppDynamics Integration v1.5	AppDynamics v4.0, 4.1
AWS CloudWatch Integration v1.6	aws-java-sdk v1.11
AWS SNS Integration v1	AWS SNS v2016-06-28
CA Nimsoft UIM Integration v1.4	CA Nimsoft UIM v8.4
CA Spectrum Integration v1.2	CA Spectrum v10.2
Cherwell Service Management Integration v1.3	Cherwell v9.3
Datadog Polling Integration v1	Datadog v2018
Datadog Webhook Integration v1.6	Datadog v5.21
Dynatrace APM Integration v1.8	Dynatrace v6.5, 7.0
Dynatrace APM Plugin Integration v1.4	Dynatrace v6.5, 7.0
Dynatrace Notification Integration v1.1	Dynatrace v6.5
Dynatrace Synthetic Integration v1.5	Dynatrace Synthetic v2017
Email Integration v2	IMAP, IMAPS, POP3, POP3S
EMC Smarts Integration v1	EMC Smarts v9.5
ExtraHop Integration v1	ExtraHop v7.3.2.1656
FluentD Integration v1.6	FluentD v0.12
Grafana Integration v6.5	Grafana v4.6 or above
HP NNMi Integration v2	HP NNMi v10.2
HP OMi Integration v1.5	HP OMi v10.1
HP OMi Plugin Integration v1.2	HP OMi v10.1
JDBC Integration v1	JDBC v4.2
JIRA Service Desk Integration v1.1	JIRA Service Desk v7.6
JIRA Software Integration v1.1	JIRA Software v7
JMS Integration v1.4	ActiveMQ v5.14, JBoss v10, WebLogic v12.0
Microsoft Azure Integration v1	Microsoft Azure Monitor v2018
Microsoft Azure Classic Integration v1	Microsoft Azure Classic v2018
Microsoft SCOM Integration v4.1	Microsoft SCOM v2012, 2016
Nagios Integration v2.5	Nagios vXI
New Relic Integration v1.7	New Relic v2016
New Relic Polling Integration v1	New Relic v2.3
Node Red Integration v1.5	Nagios Red v016, 017
Node.js Integration v1.6	Node.js v1.6
Oracle Enterprise Manager Integration v1.4	Oracle Enterprise Manager v12c, 13c
Pingdom Integration v1.5	Pingdom v2017
Remedy Integration v1	Remedy v9.1

ServiceNow Integration v2.2.4	ServiceNow vHelsinki, Istanbul, Jakarta, Kingston
SevOne Integration v1	SevOne v5.7.2.0
Slack Integration v1	Slack v3.1
SolarWinds Integration v2.4	SolarWinds v11.5, 12.2
Splunk SaaS Addon Integration v1.8	Splunk v6.5, 6.6, 7.0
Sumo Logic Integration v1	Sumo Logic v2018
VMWare vCenter Integration v2	VMWare vCenter v6.0, 6.5
VMWare vRealize Log Insight Integration v2	VMWare vRealize Log Insight v4.3
VMWare vROps Integration v2	VMWare vROps v6.6
VMWare vSphere Integration v2	VMWare vSphere v6.0, 6.5
WebSphere MQ Integration v1.6	WebSphere MQ v8
xMatters Integration v4	xMatters v5.5
Zabbix Integration v2	Zabbix v3.2
Zenoss Integration v2	Zenoss v4.2

Sizing Recommendations

The sizing recommendations below are guidelines for small, medium and large Cisco Crosswork Situation Manager systems based on input data rate and volume.

In the context of this guide, Managed Elements (MEs) are all of the components in the network infrastructure that generate and emit events:

Small

Environment	CPU	File System
• 1000 to 5000 Managed Elements (MEs) • Less than 20 users • Up to 5 integrations • Less than 20 Alerts per second	• 8 Cores • 32GB RAM • 2 x 1GB Ethernet • Physical or Virtual Server	• 1 TB Local or SAN **See retention policy

Medium

Environment	CPU	File System
• 5000 to 20,000 MEs • Between 20 and 40 users • Between 6 and 10 integrations • Between 20 and 100 Alerts per second	• 16 Cores • 64GB RAM • 2 x 1GB Ethernet • Physical or Virtual Server	• 1 TB Local or SAN **See retention policy

Large

Environment	CPU	File System
• More than 20,000 MEs • More than 40 users • More than 10 integrations • More than100 Alerts per second	• 24+ Cores • 128GB RAM • 2 x 1GB Ethernet • Physical or Virtual Server	• 1 TB Local or SAN **See retention policy

Virtualization Restrictions

Consider the following restrictions for virtual environments:

- Ideally all Moog servers (guests) should be on the same compute node (host) sharing a hypervisor or virtual machine monitor. This minimizes latency between Moog guests.
- If servers are liable to automated resource balancing (e.g. vMotion) and liable to move compute nodes, then all Moog servers should be moved at the same time. If this is not possible, then Moog servers should be constrained to movements that minimize the resulting network distance.
- If Moog servers are distributed amongst compute nodes then the network “distance” (logical hops) between the nodes should be minimized.
- Network latency between components may affect Event processing throughput. This is especially true of the core to db servers.

Shared Storage

On any shared compute platform Cisco makes the following recommendations:

- The minimum resource requirements are multiplied by at least 33% to account for shared resource usage and allocation.
- Storage latency will reduce effective throughput at the core processing layer and should be minimised within the available constraints of a SAN.
- Moog should be treated as a highly transactional system and not placed on the same compute node as other highly transactional applications that may cause SAN resource contention.
- SAN port and array port contention should be minimized
- Storage medium should be as fast as possible to minimize the transaction times to the database.

**** Retention Policy**

You can calculate the amount of disk space required for the database using the following:

(Aggregate event rate / the number of seconds per day x desired retention x average event size) / 1,000,000

- Where the aggregate event rate is across all LAMs
- Desired retention is in days
- Average event size is in Kb
- Result is in Gb.
- Some event sources have larger than average events (e.g SCOM) but generally a 2kb event size is a reasonable assumption. This 2kb base takes account of the other event/alert based storage (e.g an alerts situation membership, typical custom_info, situation room thread sizes etc.)
- So for an event rate of 10/s per LAM with 5 lams and a 400 day (13 month) retention we would have

50 x 60 x 60 x 24 x 400 x 2 / 1,000,000 = 3,456GN (3.5TB)

Pre-install Cisco Crosswork Situation Manager - RPM

Carry out the following steps to prepare a RHEL7/CentOS 7 server for installation of Cisco Crosswork Situation Manager.

Step	Action
1	Install the **EPEL** yum repository (Extra Packages for Enterprise Linux: EPEL): `yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm` Verify the creation of file `/etc/yum.repos.d/epel.repo`

2	Install the **"el7" MySQL Community** yum repository: `yum -y install http://repo.mysql.com/mysql57-community-release-el7.rpm` Verify the creation of file `/etc/yum.repos.d/mysql-community.repo`
3	Install the erlang el7 package built by RabbitMQ: `yum -y install https://github.com/rabbitmq/erlang-rpm/releases/download/v20.1.7/erlang-20.1.7-1.el7.centos.x86_64.rpm` The file is available from this page if needed: https://github.com/rabbitmq/erlang-rpm/releases/tag/v20.1.7
4	Install the **RabbitMQ** yum repository: `curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-server/script.rpm.sh | sudo bash` Verify the creation of file `/etc/yum.repos.d/rabbitmq_rabbitmq-server.repo`

5	Install the **ElasticSearch** public signing key:
	`rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch`
	Create the **ElasticSearch** yum repository by creating file `/etc/yum.repos.d/elasticsearch.repo` with contents:
	`[elasticsearch-5.x]`
	`name=Elasticsearch repository for 5.x packages`
	`baseurl=https://artifacts.elastic.co/packages/5.x/yum`
	`gpgcheck=1`
	`gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch`
	`enabled=1`
	`autorefresh=1`
	`type=rpm-md`

6	Install the nginx yum repo by creating a script (e.g.: **create_nginx_repo.sh**) containing the following contents and then executing it. e.g.: **bash create_nginx_repo.sh**
	
	`#!/bin/bash`
	
	`echo '[nginx]' > /etc/yum.repos.d/nginx.repo`
	`echo 'name=nginx repo' >> /etc/yum.repos.d/nginx.repo`
	`echo 'baseurl=http://nginx.org/packages/OS/OSRELEASE/$basearch/' >> /etc/yum.repos.d/nginx.repo`
	`echo 'gpgcheck=0' >> /etc/yum.repos.d/nginx.repo`
	`echo 'enabled=1' >> /etc/yum.repos.d/nginx.repo`
	
	`OS_VERSION=$(cat /etc/system-release)`
	`case "$OS_VERSION" in`
	`CentOS*release\ 7* )`
	`sed -i -e 's/OS/centos/' -e 's/OSRELEASE/7/' /etc/yum.repos.d/nginx.repo;;`
	`Red\ Hat*release\ 7* )`
	`sed -i -e 's/OS/rhel/' -e 's/OSRELEASE/7/' /etc/yum.repos.d/nginx.repo;;`
	`esac`

7	Ensure the **nss** and **openssl** packages are up to date by running: `yum -y update nss openssl` If these packages are already up to date then this command will exit without action

8	Create the **Moogsoft** yum repository by creating file `/etc/yum.repos.d/moogsoft-aiops.repo` with contents:
	`[moogsoft-aiops]`
	`name=moogsoft-aiops-latest`
	`baseurl=https://<login>:<password>@speedy.moogsoft.com/repo/aiops/esr`
	`enabled=1`
	`gpgcheck=0`
	`sslverify=0`
	Access to the Cisco yum repository is restricted and requires a <login> and <password>
	Please contact support@moogsoft.com regarding access

9	Install the tomcat-native package using the command below. Please provide your 'speedy' yum credentials to complete the command: `yum -y install https://<login>:<password>@speedy.moogsoft.com/offline/7/tomcat-native-1.1.34-1.el7.x86_64.rpm`
10	Set SELinux to permissive mode or disable completely. For example to set SELinux to permissive mode: `setenforce 0` If you want to disable SELinux at boot time, you can edit the file `/etc/sysconfig/selinux`.

Next step: Single Host Installation - RPM

Pre-Install Cisco Crosswork Situation Manager - Offline RPMs

If you cannot use yum to connect to package repositories outside your network from the machines where you are installing Cisco Crosswork Situation Manager, you can download tarball packages of the Cisco Crosswork Situation Manager repositories to run an offline installation. The offline repository distributions include all required packages to install Cisco Crosswork Situation Manager on a RHEL/CentOS 7 server.

These instructions guide you through the process to set up the local yum repositories so you can continue with an offline installation or upgrade. After you set up your repositories, you can continue with one of the following:

- Single Host Installation
- Distributed Installation - RPM
- Upgrade Cisco Crosswork Situation Manager

This procedure does not support a relocatable installation.

Before You Begin

- Contact Cisco Support to request the offline tarballs.

- Ensure you have root access to the system where you are installing.

Cisco Crosswork Situation Manager Installation Files

To improve download times, the distribution for the offline installation of Cisco Crosswork Situation Manager comes in two separate archives:

- A "BASE" repository containing the dependent packages to install Cisco Crosswork Situation Manager for RHEL/CentOS 7. The base package follows the following naming convention:

```
<date/timestamp>-MoogsoftBASE7_offline_repo.tar.gz
Example: 2018-09-26-1537962719-MoogsoftBASE7_offline_repo.tar.gz
```

- An "ESR" repository that contains the standard Cisco Crosswork Situation Manager RPMs and ancillary packages (Tomcat, RabbitMQ, JRE etc)

```
ESR: <date/timestamp>-MoogsoftESR_<version>_offline_repo.tar.gz
Example: 2018-09-26-1537962719-MoogsoftESR_7.0.0_offline_repo.tar.gz
```

Download Installation Files

Before you can set up the local yum repositories, you need to download installation files from a machine connected to the internet. Then copy the installation files to a directory on the target system. The examples use `/home` for the installation file directory.

1. Use the links from Cisco support to download the Cisco Crosswork Situation Manager installation files on an internet-connected host and copy them to the target offline system.
2. Download the MySQL-Community packages on an internet-connected host and copy them to the target offline system.

   For example on RHEL7/CentOS7:

```
curl -L -O https://repo.mysql.com/yum/mysql-5.7-community/el/7/x86_64
/mysql-community-libs-5.7.22-1.el7.x86_64.rpm
curl -L -O https://repo.mysql.com/yum/mysql-5.7-community/el/7/x86_64
/mysql-community-libs-compat-5.7.22-1.el7.x86_64.rpm
curl -L -O https://repo.mysql.com/yum/mysql-5.7-community/el/7/x86_64
/mysql-community-server-5.7.22-1.el7.x86_64.rpm
curl -L -O https://repo.mysql.com/yum/mysql-5.7-community/el/7/x86_64
/mysql-community-common-5.7.22-1.el7.x86_64.rpm
curl -L -O https://repo.mysql.com/yum/mysql-5.7-community/el/7/x86_64
/mysql-community-client-5.7.22-1.el7.x86_64.rpm
```

3. Download the mysql-connector-5.1.45 jar file:

```
curl -L -O https://dev.mysql.com/get/Downloads/Connector-J/mysql-
connector-java-5.1.45.tar.gz
```

Prepare the Local yum Repositories

The following procedure describes how to create local yum repositories to house the installation packages for Cisco Crosswork Situation Manager that you downloaded. If you are running a distributed installation, follow this procedure on each machine where you run Cisco components.

1. Create directories that will house the repositories. For example:

```
sudo mkdir -p /media/localRPM/BASE/

sudo mkdir -p /media/localRPM/ESR/
```

2. Extract the two packages into separate directories. For example:

```
tar xzf /home/*-MoogsoftBASE7_offline_repo.tar.gz -C /media/localRPM
/BASE/

tar xzf /home/*-MoogsoftESR_7.0.0_offline_repo.tar.gz -C /media
/localRPM/ESR/
```

3. Move the existing `/etc/yum.repos.d.` directory to create a backup:

```
mv /etc/yum.repos.d /etc/yum.repos.d-backup
```

4. Creating an empty /etc/yum.repos.d directory

```
 mkdir /etc/yum.repos.d
```

5. Create a `local.repo` for yum:

```
vi /etc/yum.repos.d/local.repo
```

6. Edit the contents of `local.repo`. Verify the path for the base and for the baseurl points to the directories you created.

```
[BASE]
name=MoogCentOS-$releasever - MoogRPM
baseurl=file:///media/localRPM/BASE/RHEL
gpgcheck=0
enabled=1
[ESR]
name=MoogCentOS-$releasever - MoogRPM
baseurl=file:///media/localRPM/ESR/RHEL
gpgcheck=0
enabled=1
```

7. Clean the yum cache to remove cached files from any enabled repository:

```
yum clean all
```

8. Verify yum detects the the newly created local repositories:

```
yum info "moogsoft-*"
```

```
Available Packages
Arch        : x86_64
Version     : 7.0.0
Release     : 8
Size        : 76 M
Repo        : ESR
Summary     : Algorithmic Intelligence for IT Operations
URL         : https://www.moogsoft.com
License     : Proprietary
Description : Cisco Crosswork Situation Manager (6.5.0) -
Build: 142 - (Revision:
```

The results should include the following packages:

```
Name        : moogsoft-db
Name        : moogsoft-integrations
Name        : moogsoft-integrations-ui
Name        : moogsoft-mooms
Name        : moogsoft-search
Name        : moogsoft-server
Name        : moogsoft-ui
Name        : moogsoft-utils
Name        : moogsoft-common
```

9. Update the system if needed. Create an exclusion list if you prefer to keep certain packages away from update:

```
yum update
```

10. Install the MySQL RPMs:

```
yum install mysql-community-*5.7.22*.rpm
```

If yum reports errors regarding mariadb, replace mariadb with the equivalent mysql package. The following script removes mariadb-server and replaces it with mysql-community-server:

```
echo "remove mariadb-server" >> /tmp/mysql.libs
echo "install mysql-community-libs-compat-5.7.22" >> /tmp/mysql.libs
echo "install mysql-community-server-5.7.22" >> /tmp/mysql.libs
echo "run" >> /tmp/mysql.libs
cat /tmp/mysql.libs | yum shell -y
```

Your local yum repos are ready. Now you can proceed with your offline install or upgrade. See the following topics:

- Single Host Installation - RPM

Single Host Installation - RPM

This topic guides you through the installation process for Cisco Crosswork Situation Manager using RPM via yum.

Before You Begin

Before you install Cisco Crosswork Situation Manager, ensure you have met the

- following requirements: You have root access to the machine where you are installing.
- You have completed the pre-installation procedures for your environment:
  - Pre-install Cisco Crosswork Situation Manager - RPM
  - Pre-install Offline

Install Cisco Crosswork Situation Manager

To install Cisco Crosswork Situation Manager on a single host machine:

1. Download the RPMs and install them into `$MOOGSOFT_HOME`:

```
yum -y install moogsoft-db \
                        moogsoft-common \
            moogsoft-integrations \
            moogsoft-integrations-ui \
            moogsoft-mooms \
            moogsoft-search \
            moogsoft-server \
            moogsoft-ui \
            moogsoft-utils
```

2. Set the environment variables required for Cisco Crosswork Situation Manager.

   Before you can create the APPSERVER_HOME variable, you must create the target apache-tomcat directory:

```
mkdir /usr/share/apache-tomcat
```

   Then, if you are using the default Bash shell, you can set the varibles in `~/.bashrc`:

```
vi ~/.bashrc
```

   Add the following by copy/paste line by line:

```
export MOOGSOFT_HOME=/usr/share/moogsoft
export JAVA_HOME=/usr/java/latest
export APPSERVER_HOME=/usr/share/apache-tomcat
export PATH=$PATH:$MOOGSOFT_HOME/bin:$MOOGSOFT_HOME/bin/utils
```

When entering the environment variable in the bashrc file, ensure each export command is on separate single line and that it does not wrap.

3. Start a new Bash shell, or source the `.bashrc` file as follows:

```
source ~/.bashrc
```

4. If you have not already downloaded the MySQL connector as part of Pre-Install Cisco Crosswork Situation Manager - Offline RPMs, download the mysql-connector-5.1.45 jar file :

```
curl -L -O https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.45.tar.gz
```

5. Extract, rename and move the file to $MOOGSOFT_HOME/lib/cots/nonDist/ as follows:

```
tar --strip-components 1 -xvf mysql-connector-java-5.1.45.tar.gz -C $MOOGSOFT_HOME/lib/cots/nonDist/ mysql-connector-java-5.1.45/mysql-connector-java-5.1.45-bin.jar --transform 's/-bin//'
```

6. Initialize the installation:

```
$MOOGSOFT_HOME/bin/utils/moog_init.sh -I MY_ZONE -u root
```

   - The default password for MySQL is empty, so press Enter when prompted for the MySQL 'root' password. If you created a separate user in the database for Cisco Crosswork Situation Manager, use the credentals for the user you created.
   - Follow the instructions and confirm prompts about the hostname.

7. Start moogfarmd:

```
service moogfarmd start
```

Verify your installation following the steps in Post Install Validation.
Follow the Troubleshooting guide to resolve any issues.

Distributed Installation - RPM

You can install the different components of Cisco Crosswork Situation Manager to multiple machines in order to distribute the workload. Dividing the workload for Cisco Crosswork Situation Manager components between multiple machines can provide performance improvement, but it does not provide redundancy or failover. For those features, see High Availability.

Distributed Install Procedure

1. Choose which host each component should be installed on, and run one or more of the following commands on the selected hosts as required by your distributed architecture.

```
yum install moogsoft-db
yum install moogsoft-integrations
yum install moogsoft-integrations-ui
yum install moogsoft-mooms
yum install moogsoft-search
yum install moogsoft-server
yum install moogsoft-ui
yum install moogsoft-utils
yum install moogsoft-common
```

2. Set the environment variables (via `.bashrc`) on each host with the following commands:

```
export MOOGSOFT_HOME=/usr/share/moogsoft
export JAVA_HOME=/usr/java/latest
export APPSERVER_HOME=/usr/share/apache-tomcat
export PATH=$PATH:$MOOGSOFT_HOME/bin:$MOOGSOFT_HOME/bin/utils
```

3. Download the MySQL connector (`mysql-connector-5.1.45 jar`). For example:

```
curl -L -O https://dev.mysql.com/get/Downloads/Connector-J/mysql-
connector-java-5.1.45.tar.gz
```

4. Extract the MySQL connector, rename it `mysql-connector-java-5.1.45.jar` and coypy it to `$MOOGSOFT_HOME/lib/cots/nonDist/`. For example:

```
tar --strip-components 1 -xvf mysql-connector-java-5.1.45.tar.gz \
    -C $MOOGSOFT_HOME/lib/cots/nonDist/ mysql-connector-java-5.1.45
/mysql-connector-java-5.1.45-bin.jar \
    --transform 's/-bin//'
```

5. Initialize the installation as described below.

Initializing a distributed installation

Each installed package has its own `init` script (for example `moog_init_ui.sh`). The individual package `init` scripts make configuring relevant properties more straightforward, such as those in `$MOOGSOFT_HOME/config/system.conf`.
For example, `moog_init_ui.sh` includes the following optional arguments:

- `-d` Configure `system.conf` with the hostname and port of the MySQL server
- `-m` Configure `system.conf` with the hostname and port of the RabbitMQ (MooMS) server
- `-s` Configure `system.conf` with the hostname and port of the search Elasticsearch server
- `-z` Configure `system.conf` and other UI files with the Zone name and RabbitMQ vhost

Examples of initializations for distributed installations

SERVER1 and **SERVER2** are example hostnames. **MY_ZONE** is an example MooMS Zone name.

Example 1: `moogsoft-db` on SERVER1 and all other RPMs on SERVER2

The **db** package can be initialized with basic options but most other packages then need to know its location for their initialization.

- Initialize the database on **SERVER1**:

```
moog_init_db.sh -Iu root
```

- On **SERVER1**, connect to mysql as the root user and grant all access on the **moogdb** and **moog_reference** databases to the **ermintrud** user on **SERVER2**:

```
GRANT ALL ON moogdb.* TO ermintrude@'SERVER2' IDENTIFIED BY 'm00';
GRANT ALL ON moog_reference.* TO ermintrude@'SERVER2' IDENTIFIED BY
'm00';
```

- Initialize the remaining components on **SERVER2**:

```
moog_init_mooms.sh -pz MY_ZONE
moog_init_lams.sh -bz MY_ZONE -d SERVER1:3306
moog_init_search.sh -d SERVER1:3306
moog_init_server.sh -bz MY_ZONE -d SERVER1:3306
moog_init_ui.sh -twxnfz MY_ZONE -d SERVER1:3306
```

Example 2: `moogsoft-ui` on SERVER1 and all other RPMs on SERVER2

On initialization, the **ui** package needs to know the following:

- The host:port of the database
- The host:port of MooMS broker
- The host:port of Elasticsearch
- The host:port of MooMS admin console
- The MooMS Zone to connect to

- Initialize the components on **SERVER2**:

```
moog_init_db.sh -Iu root
moog_init_mooms.sh -pz MY_ZONE
moog_init_server.sh -b
moog_init_search.sh
moog_init_lams.sh -bz MY_ZONE
```

- On **SERVER2**, connect to mysql as the root user and grant all access on the **moogdb** and **moog_reference** databases to the **ermintrud** user on **SERVER1**:

```
GRANT ALL ON moogdb.* TO ermintrude@'SERVER1' IDENTIFIED BY 'm00';
GRANT ALL ON moog_reference.* TO ermintrude@'SERVER1' IDENTIFIED BY
'm00';
```

- By default, Elasticsearch listens only on localhost so must be made to listen on the outbound interface. On **SERVER2**, edit the /etc /elasticsearch/elasticsearch.yml file and set the following property:

```
http.host: 0.0.0.0
```

- On **SERVER2**, restart the elasticsearch service:

```
service elasticsearch restart
```

- Finally, run this command on **SERVER1**:

```
moog_init_ui.sh -twxfz MY_ZONE -c SERVER2:15672 -d SERVER2:3306 -m
SERVER2:5672 -s SERVER2:9200
```

Example 3: `moogsoft-server` on SERVER1 and all other RPMs on SERVER2

On initialization, the **server** package needs to know the following:

- The host:port of the database
- The host:port of MooMS broker
- The MooMS Zone to connect to

- Initialize the components on **SERVER2**:

```
moog_init_db.sh -Iu root
moog_init_mooms.sh -pz MY_ZONE
moog_init_lams.sh -bz MY_ZONE
moog_init_search.sh
moog_init_ui.sh -twxnfz MY_ZONE
```

- On **SERVER2**, connect to mysql as the root user and grant all access on the **moogdb** and **moog_reference** databases to the **ermintrud** user on **SERVER1**:

```
GRANT ALL ON moogdb.* TO ermintrude@'SERVER1' IDENTIFIED BY 'm00';
GRANT ALL ON moog_reference.* TO ermintrude@'SERVER1' IDENTIFIED BY
'm00';
```

- Run this command on **SERVER1**:

```
moog_init_server.sh -bez MY_ZONE -d SERVER2:3306 -m SERVER2:5672
```

> The moog_farmd service is not started or stopped by the `init` script so this should be done manually.

Example 4: `moogsoft-lams` on SERVER1 and all other RPMs on SERVER2

On initialization, the **lam** package needs to know the following:

- The host:port of the database
- The host:port of MooMS broker
- The MooMS Zone to connect to

- Initialize the components on **SERVER2**:

```
moog_init_db.sh -Iu root
moog_init_mooms.sh -pz MY_ZONE
moog_init_server.sh -bz MY_ZONE
moog_init_search.sh
moog_init_ui.sh -twxnfz MY_ZONE
```

- On **SERVER2**, connect to mysql as the root user and grant all access on the **moogdb** and **moog_reference** databases to the **ermintrud** user on **SERVER1**:

```
GRANT ALL ON moogdb.* TO ermintrude@'SERVER1' IDENTIFIED BY 'm00';
GRANT ALL ON moog_reference.* TO ermintrude@'SERVER1' IDENTIFIED BY
'm00';
```

- Run this command on **SERVER1**:

```
moog_init_lams.sh -bz MY_ZONE -d SERVER2:3306 -m SERVER2:5672
```

Upgrading a distributed installation

Regardless of how many RPMs have been installed on a single host, all of the packages must be upgraded at the same time on that host

Similarly, different versions of packages running at the same time are not supported in a distributed environment, so all hosts should be upgraded to the same package version at the same time

The following example demonstrates the type of error which `yum` will produce if only one out of two packages is upgraded. If `moogsoft-server` and `moogsoft-integrations` have been installed on a single host, running an upgrade for only `moogsoft-server` (via `yum update moogsoft-server`) will generate errors from `yum` similar to those below:

```
Transaction Check Error:
  file /usr/share/moogsoft/lib/events_analyser.jar from install of
moogsoft-server-6.5.0-436.x86_64 conflicts with file from package moogsoft-
integrations-6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/farmd_cntl.jar from install of moogsoft-
server-6.5.0-436.x86_64 conflicts with file from package moogsoft-
integrations-6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/ha_cntl.jar from install of moogsoft-server-
6.5.0-436.x86_64 conflicts with file from package moogsoft-integrations-
6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/moobot.jar from install of moogsoft-server-
6.5.0-436.x86_64 conflicts with file from package moogsoft-integrations-
6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/moog_add_alert_custom_field.jar from
install of moogsoft-server-6.5.0-436.x86_64 conflicts with file from
package moogsoft-integrations-6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/moog_add_sitn_custom_field.jar from install
of moogsoft-server-6.5.0-436.x86_64 conflicts with file from package
moogsoft-integrations-6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/moog_config_reader.jar from install of
moogsoft-server-6.5.0-436.x86_64 conflicts with file from package moogsoft-
integrations-6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/moog_encryptor.jar from install of moogsoft-
server-6.5.0-436.x86_64 conflicts with file from package moogsoft-
integrations-6.5.0-435.x86_64
  file /usr/share/moogsoft/lib/moog_farmd.jar from install of moogsoft-
server-6.5.0-436.x86_64 conflicts with file from package moogsoft-
integrations-6.5.0-435.x86_64
...
```

The supported upgrade path (for the above example) runs the following command:

```
yum update moogsoft-server moogsoft-integrations
```

You must run the relevant `init` scripts so the updated components know where the rest of the packages are located.

Problems?: Troubleshooting

Single Host Installation for Non-root

This topic covers the procedure to install Cisco Crosswork Situation Manager for the following scenarios:

- You can perform the installation as a user other than `root`. The installation steps also work for the `root` user. Be sure to perform all steps as the same user.
- You can install Cisco Crosswork Situation Manager to the directory of your choice.
- You do not need to use a package manager (RPM) to install Cisco Crosswork Situation Manager.

Before You Begin

Before you run the Cisco Crosswork Situation Manager installation, perform the following verification and preparation steps:

1. Choose a CentOS 7 / RHEL 7 system.

> These instructions are for a new installation and assume you have not previously installed Cisco Crosswork Situation Manager. If you have defined Moogsoft environment variables such as `$MOOGSOFT_HOME` for the installation user, remove the environment variables before continuing the installation.

2. Identify the Linux user you want to use for installation. Use the same user credentials later if for any reason you need to start or stop Cisco Crosswork Situation Manager processes.
3. Optionally set the ulimit maximum for open files and max user processes for the installation user. For example, on a busy system you could increase both to 65535. Setting ulimit values requires `root` permissions.
4. Choose a working directory to run your installation. The examples in this document use `/home/admin` as the working directory. The installation directory requires a minimum of 7GB. If you are also storing the MySQL database in the installation directory, you will require more space to allow the growth of log files and storage of artifacts in the database and Elasticsearch.
5. Download the Cisco Crosswork Situation Manager non-root installer. You can download this via a web browser from https://speedy.moogsoft.com/installer/ a nd use the yum user credentials provided by Cisco support. Alternatively you can update the following command with your user credentials to download the installer:

```
curl -L -O "https://<username>:<password>@speedy.moogsoft.com
/installer/moogsoft-aiops-7.0.0.tgz"
```

6. Unzip and untar the distribution archive in your working directory:

```
tar -xf moogsoft-aiops-7.0.0.tgz
```

The distribution archive moogsoft-aiops-<version>.tgz contains:

- a README.txt file.
- the installation script: moogsoft-aiops-install-7.0.0.sh.
- a Cisco Crosswork Situation Manager archive: moogsoft-aiops-dist-7.0.0.tgz.
- an integrations archive: moogsoft-aiops-integrations-7.0.0.tgz.

7. Download the MySQL connector v5.1.45 (mysql-connector-java-5.1.45) and copy it to your working directory. For example:

```
curl -L -O https://dev.mysql.com/get/Downloads/Connector-J/mysql-
connector-java-5.1.45.tar.gz
```

8. Install MySQL mysql-community-server-5.7.22. For example to install to your working directory:
   - Download the mysql-community-server-5.7.22 distribution package to your working directory:

```
curl -L -O https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-
5.7.22-el7-x86_64.tar.gz
```

   - Unzip and untar the MySQL distribution package:

```
tar -xf mysql-5.7.22-el7-x86_64.tar.gz
```

   - Add the MySQL binary path to your `$PATH` environment variable.

```
echo "export PATH=$PATH:<working-directory>/mysql-5.7.22-el7-
x86_64/bin" >> ~/.bashrc && \
source ~/.bashrc
```

   For instance, using the `/home/admin` working directory:

```
```

```
echo "export PATH=$PATH:/home/admin/mysql-5.7.22-el7-x86_64/bin"
>> ~/.bashrc && \
source ~/.bashrc
```

9. Install Kernel Asynchronous I/O (AIO) Support for Linux. For example:

```
mkdir -p ~/install/libraries/ && cd ~/install/libraries/ && \
curl -L -O http://mirror.centos.org/centos/7/os/x86_64/Packages
/libaio-0.3.109-13.el7.x86_64.rpm && \
rpm2cpio ./libaio-0.3.109-13.el7.x86_64.rpm | cpio -idmv && \
rm -f ./libaio-0.3.109-13.el7.x86_64.rpm && \
rm -f ~/install/libraries/lib64/libaio.so.1 && \
ln -s ~/install/libraries/lib64/libaio.so.1.0.1 ~/install/libraries
/lib64/libaio.so.1 && \
echo "export LD_LIBRARY_PATH=`pwd`/lib64:\$LD_LIBRARY_PATH" >> ~/.
bashrc && \
source ~/.bashrc && \
cd -
```

10. Navigate to your working directory and install libgfortran v4.4.6 or later. For example:

```
mkdir -p ~/install/libraries/ && cd ~/install/libraries/ && \
curl -L -O http://mirror.centos.org/centos/7/os/x86_64/Packages
/libquadmath-4.8.5-28.el7.x86_64.rpm && \
rpm2cpio ./libquadmath-4.8.5-28.el7.x86_64.rpm | cpio -idmv && \
rm ./libquadmath-4.8.5-28.el7.x86_64.rpm && \
curl -L -O http://mirror.centos.org/centos/7/os/x86_64/Packages
/libgfortran-4.8.5-28.el7.x86_64.rpm && \
rpm2cpio ./libgfortran-4.8.5-28.el7.x86_64.rpm | cpio -idmv && \
rm ./libgfortran-4.8.5-28.el7.x86_64.rpm && \
echo "export LD_LIBRARY_PATH=`pwd`/usr/lib64:\$LD_LIBRARY_PATH" >> ~/.
bashrc && \
source ~/.bashrc && \
cd -
```

11. Verify ports 8443 and 8080 are open. If you are installing as `root`, Verify ports 443 and 80 are open.
12. Verify your system is running OpenSSL v1.02 or later.

Install Cisco Crosswork Situation Manager

1 Execute the installation script `moogsoft-aiops-install-7.0.0.sh` in your working directory to install Cisco Crosswork Situation Manager.

The script guides you through the installation process and lets you choose the installation directory, by default <working-directory> /moogsoft.

```
bash moogsoft-aiops-install-7.0.0.sh
```

2 As a convenience, set the `$MOOGSOFT_HOME` environment variable to point to your installation directory. Additionally, add `$MOOGSOFT_HOME/bin/utils` to the path. For example:

```
echo "export MOOGSOFT_HOME=/home/admin/moogsoft" >> ~/.bashrc
echo "export PATH=$PATH:$MOOGSOFT_HOME/bin/utils" >> ~/.bashrc && \
source ~/.bashrc
```

Initialize Cisco Crosswork Situation Manager

After you finish the installation process, initialize Cisco Crosswork Situation Manager as follows:

1. Configure the toolrunner to execute locally. Edit `$MOOGSOFT_HOME/config/servlets.conf`.
2. Uncomment the `execute_locally` property, and value set the value to 'true' as follows. Take special note of the initial comma:

```
          , execute_locally: true
```

3. Execute the initialization script startup and bootstrap your system:

```
$MOOGSOFT_HOME/bin/utils/moog_init.sh -I <zone_name> -u root
```

   The zone name sets up a virtual host for the message bus. If you have multiple systems sharing the same bus, use a different zone name for each. For example:

```
moog_init.sh -I MoogsoftAIOps -u root
```

   The script prompts you to accept the End User License Agreement (EULA) and guides you through the initialization process.

   The script initializes all components on the current host, loads the MySQL schemas using the MySQL user.

Unattended Installation Example

The moog_init script provides the capability to run a quiet install and to automatically accept the EULA. This means you can write a bash script to automatically execute both the installation script and the initialization script on the same host. For example:

```
/home/admin/moogsoft-aiops-install-7.0.0.sh \
 -d /home/admin/moogsoft &&

/home/admin/moogsoft/bin/utils/moog_init.sh \
 -qI MoogsoftAIOps -p MySQLpasswd -u root --accept-eula
```

Run `moog_init.sh -h` for a description of all the flags.

Post-Initialization

Verify your installation by following the steps in Post Install Validation.

You can use the process_cntl utility to start, stop, and check the status of the processes. See Control Cisco Crosswork Situation Manager Processes.

Follow the Troubleshooting guide to resolve any issues.

Distributed Installation for Non-root

You can install the different of Cisco Crosswork Situation Manager packages on multiple machines in order to distribute the workload. Dividing the workload for Cisco Crosswork Situation Manager components between multiple machines can a provide performance improvement, but it does not provide redundancy or failover. For those features, see High Availability.

This topic guides you through the most common distributed installation scenario, which is to run the MySQL database on one host and all other processes on a separate host.

Before You Begin

Before you run the Cisco Crosswork Situation Manager installation, perform the following verification and preparation steps:

- Identify the Centos 7 hosts where you plan run your distributed installation. The example uses server1 for the database host and server2 for the host for all other processes.
- Identify the Linux user you want to use for installation. Use the same user credentials later if for any reason you need to start or stop Cisco Crosswork Situation Manager processes.
- Optionally set the ulimit maximum for open files and max user processes for the installation user. For example, on a busy system you could increase both to 65535. Setting ulimit values requires root permissions.
- Choose a working directory to run your installation. The examples in this document use the current user's home folder, `~` , as the working directory. The installation directory requires a minimum of 7GB. If you are also storing the MySQL database in the installation directory, you'll require more space to allow the growth of log files and storage of artefacts in the database and ElasticSearch.
- Verify ports 8443 and 8080 are open. If you are installing as root, Verify ports 443 and 80 are open.
- Ensure both servers can communicate using the MySQL port: 3306 by default.
- Verify your system is running OpenSSL v1.02 or later.

Install Cisco Crosswork Situation Manager and MySQL

Install Cisco Crosswork Situation Manager and MySQL on both hosts, for example server1 and server2. The database server uses some components from Cisco Crosswork Situation Manager and Cisco Crosswork Situation Manager uses the database client.

Perform the following installation steps on both hosts unless otherwise noted:

1. Download the Cisco Crosswork Situation Manager non-root installer. You can download this via a web browser from https://speedy.moogsoft.com/installer/ a nd use the yum user credentials provided by Cisco support. Alternatively you can update the following command with your user credentials to download the installer:

   ```
   curl -L -O "https://<username>:<password>@speedy.moogsoft.com
   /installer/moogsoft-aiops-7.0.0-19745.tgz"
   ```

2. Unzip and untar the distribution archive in your working directory:

   ```
   tar -xf moogsoft-aiops-7.0.0-19745.tgz
   ```

3. Download the MySQL connector v5.1.45 (mysql-connector-java-5.1.45) and copy it to your working directory. For example:

   ```
   curl -L -O https://dev.mysql.com/get/Downloads/Connector-J/mysql-
   connector-java-5.1.45.tar.gz
   ```

4. Install MySQL mysql-community-server-5.7.22. For example to install to your working directory:
   - Download the mysql-community-server-5.7.22 distribution package to your working directory:

     ```
     curl -L -O https://dev.mysql.com/get/Downloads/MySQL-5.7/mysql-
     5.7.22-el7-x86_64.tar.gz
     ```

   - Unzip and untar the MySQL distribution package:

     ```
     tar -xf mysql-5.7.22-el7-x86_64.tar.gz
     ```

   - Add the MySQL binary path to your `$PATH` environment variable. For instance, using the admin user's home folder:

```
echo "export PATH=$PATH:~/mysql-5.7.22-el7-x86_64/bin" >> ~/.
bashrc && \
source ~/.bashrc
```

5. Install Kernel Asynchronous I/O (AIO) Support for Linux. For example:

```
mkdir -p ~/install/libraries/ && cd ~/install/libraries/ && \
curl -L -O http://mirror.centos.org/centos/7/os/x86_64/Packages
/libaio-0.3.109-13.el7.x86_64.rpm && \
rpm2cpio ./libaio-0.3.109-13.el7.x86_64.rpm | cpio -idmv && \
rm -f ./libaio-0.3.109-13.el7.x86_64.rpm && \
rm -f ~/install/libraries/lib64/libaio.so.1 && \
ln -s ~/install/libraries/lib64/libaio.so.1.0.1 ~/install/libraries
/lib64/libaio.so.1 && \
echo "export LD_LIBRARY_PATH=`pwd`/lib64:\$LD_LIBRARY_PATH" >> ~/.
bashrc && \
source ~/.bashrc && \
cd -
```

6. Install libgfortran v4.4.6 or later on the host that runs the Cisco Crosswork Situation Manager components; not the database host. For example on server2:

```
mkdir -p ~/install/libraries/ && cd ~/install/libraries/ && \
curl -L -O http://mirror.centos.org/centos/7/os/x86_64/Packages
/libquadmath-4.8.5-28.el7.x86_64.rpm && \
rpm2cpio ./libquadmath-4.8.5-28.el7.x86_64.rpm | cpio -idmv && \
rm ./libquadmath-4.8.5-28.el7.x86_64.rpm && \
curl -L -O http://mirror.centos.org/centos/7/os/x86_64/Packages
/libgfortran-4.8.5-28.el7.x86_64.rpm && \
rpm2cpio ./libgfortran-4.8.5-28.el7.x86_64.rpm | cpio -idmv && \
rm ./libgfortran-4.8.5-28.el7.x86_64.rpm && \
echo "export LD_LIBRARY_PATH=`pwd`/usr/lib64:\$LD_LIBRARY_PATH" >> ~/.
bashrc && \
source ~/.bashrc && \
cd -
```

7. Execute the installation script `moogsoft-aiops-install-7.0.0.sh` in your working directory to install Cisco Crosswork Situation Manager.

```
bash moogsoft-aiops-install-7.0.0.sh
```

The script guides you through the installation process and lets you choose the installation directory, by default `<working-directory>/moogsoft`.

8. Set the `$MOOGSOFT_HOME` environment variable to point to your installation directory. For example:

```
echo "export MOOGSOFT_HOME=/home/admin/moogsoft" >> ~/.bashrc
echo "export PATH=$PATH:$MOOGSOFT_HOME/bin/utils" >> ~/.bashrc && \
source ~/.bashrc
```

Initialize the Database Server

Initialize the database server first so it will be available for dependent processes when they start up.

To set up the database, execute the database initialization script `moog_init_db.sh` on the database host. For example on server1:

```
$MOOGSOFT_HOME/bin/utils/moog_init_db.sh -Iu root
```

Add mysql 'grants' to allow **SERVER2** to communicate with the **SERVER1** MySQL instance (please substitute **<SERVER2_IP_ADDRESS>** with the IP address of **SERVER2**):

```
mysql -u root -S $MOOGSOFT_HOME/cots/mysql/var/lib/mysql/mysql.sock <<<
"GRANT ALL PRIVILEGES ON *.* TO 'root'@'<SERVER2_IP_ADDRESS>' IDENTIFIED
BY '' with grant option;"
mysql -u root -S $MOOGSOFT_HOME/cots/mysql/var/lib/mysql/mysql.sock <<<
"GRANT ALL ON moogdb.* TO 'ermintrude'@'<SERVER2_IP_ADDRESS>' IDENTIFIED
BY 'm00'; \

GRANT ALL ON moog_reference.* TO ermintrude@'<SERVER2_IP_ADDRESS>'
IDENTIFIED BY 'm00'; \

GRANT ALL ON historic_moogdb.* TO 'ermintrude'@'<SERVER2_IP_ADDRESS>'
IDENTIFIED BY 'm00';"
mysql -u root -S $MOOGSOFT_HOME/cots/mysql/var/lib/mysql/mysql.sock <<<
"GRANT ALL ON moogdb.* TO 'root'@'<SERVER2_IP_ADDRESS>' IDENTIFIED BY ''; \

GRANT ALL ON moog_reference.* TO root@'<SERVER2_IP_ADDRESS>' IDENTIFIED BY
'';\

GRANT ALL ON historic_moogdb.* TO root@'<SERVER2_IP_ADDRESS>' IDENTIFIED
BY '';"
```

Initialize Cisco Crosswork Situation Manager

After the database is available, you can initialize the remaining Cisco Crosswork Situation Manager processes. For example on host2:

1. Configure the toolrunner to execute locally. Edit `$MOOGSOFT_HOME/config/servlets.conf`.
2. Uncomment the `execute_locally` property, and value set the value to 'true' as follows. Take special note of the initial comma and the absence of the trailing comma.

   ```
   , execute_locally: true
   ```

3. Execute the Cisco Crosswork Situation Manager initialization script, `moog_init.sh`, to set up the remaining Cisco Crosswork Situation Manager processes.

   ```
   $MOOGSOFT_HOME/bin/utils/moog_init.sh -qI MoogsoftAIOps -u root --
   accept-eula -d <server1_ip>:3306
   ```

   Where <server1_ip> is the IP address of the database server.
4. Run the following command to start moogfarmd:

```
$MOOGSOFT_HOME/bin/utils/process_cntl moog_farmd start
```

After moogfarmd starts, login to Cisco Crosswork Situation Manager at the following URL: **https://<server1_ip>:8443**

Post Install Validation

After an installation or an upgrade it is important to run the relevant validator script (after the required install/upgrade steps have been followed and `init` scripts run, etc.).

Main Cisco Crosswork Situation Manager server validation (moogfarmd, LAMs etc)

```
$MOOGSOFT_HOME/bin/utils/moog_install_validator.sh
```

Errors from the moog_install_vallidator.sh referencing Pak_docs.html can be ignored.

Apache-Tomcat webapp validation

```
$MOOGSOFT_HOME/bin/utils/tomcat_install_validator.sh
```

MySQL schema validation

```
$MOOGSOFT_HOME/bin/utils/moog_db_validator.sh
```

If you have any issues see Troubleshooting.

Go to the Operator Guide.

Upgrade - RPM Deployments

This guide provides instruction on how to upgrade to Cisco Crosswork Situation Manager ESR v7.0.0. You must upgrade to either v6.4.0, or v6.5.0 before you upgrade to v7.0.0.

For instructions on how to upgrade from a supported version, refer to the following topics:

For instructions on how to upgrade other ESR or Edge Releases, see Cisco Crosswork Situation Manager Releases.

The steps below summarize the general steps to upgrade from one version of Cisco Crosswork Situation Manager to another.

1. Back up your existing system so that you can revert changes if needed.
2. Stop Services and Processes so you can swap new upgraded libraries.
3. Upgrade Cisco Crosswork Situation Manager by installing via the new RPMs and updating configuration files.
4. Upgrade the database to include schema and data changes.
5. Upgrade Apache Tomcat webapps.
6. Restart services and processes before running the updated version of Cisco Crosswork Situation Manager.
7. Run the verification script to ensure the upgrade was successful.

Refer to the relevant upgrade pages for detailed upgrade procedures.

Upgrade v6.4.0 or V6.4.0.x to v7.0.0

This topic describes the upgrade procedure of RPM deployments from Cisco Crosswork Situation Manager v6.4.0 or v6.4.0.x to Cisco Crosswork Situation Manager v7.0.0. Refer to Upgr ade Cisco Crosswork Situation Manager - RPM Deployments for general information and links to upgrades for other versions.

The following steps outline this workflow:

Before You Begin

Red Hat Enterprise Linux 6/CentOS 6 (RHEL6) is not supported for Cisco Crosswork Situation Manager v7.0.0. See Deprecation Notice: RHEL 6. If your previous installation runs on RHEL6, see the Upgrade - Migrate from RHEL6 to RHEL7 guide before carrying out this migration

Cisco Crosswork Situation Manager no longer requires the moogsoft-eula RPM package. You accept the End User License Agreement (EULA) when you run one of the `moog_init` commands during the upgrade process. You must accept the EULA to continue with the upgrade.

To prepare for an offline upgrade, where the Cisco Crosswork Situation Manager packages reside in a 'local' yum repository, follow steps 1-8 in section "Prepare the Local yum Repositories" in the Pre-Install Cisco Crosswork Situation Manager - Offline RPMs guide. Then follow this guide to perform the upgrade. Take into account any step-specific notes regarding Offline upgrades.

Back Up the Existing System

To back up the exsting system:

1. Back up `$MOOGSOFT_HOME`.
2. Take a snapshot (for VMs).
3. Back up MySQL.

Stop Services and Processes

1. Stop moogfarmd:

    ```
    service moogfarmd stop
    ```

    Alternatively, if the moogfarmd instance was started via the UI, you should stop it via the UI at this point.
2. Ensure no more farmd processes are running with this command:

    ```
    kill -9 $(ps -ef | grep java | grep farm | awk '{print $2}') 2>/dev
    /null
    ```

3. Stop Apache Tomcat:

    ```
    service apache-tomcat stop
    ```

4. Stop the LAMs. You can do this via their service scripts:

    ```
    service <lam_service_name> stop
    ```

Alternatively, stop their running processes using the command:

```
kill <pid>
```

5. Ensure all the above processes have been stopped before continuing:

```
service <service_name> status
```

6. Disable the `events_analyser` from running during the upgrade process:

    1. Run the following command to comment out the relevant lines in `crontab`:

```
(crontab -l | sed -e 's/^\(.*events_analyser.*\)$/#\1/') |
crontab -
```

    2. Run the following command to stop any running `events_analyser` processes:

```
ps -ef | grep java | egrep 'events_analyser' | awk '{print $2}'
| xargs kill 2>/dev/null
```

Remove the logrotate cronjob

The `logrotate` utility is no longer used by Cisco Crosswork Situation Manager and is replaced by Log4j. Run the following command to remove the logrotate cronjob entry:

```
(crontab -l | grep -v logrotate) | crontab -
```

Prepare MySQL for the Upgrade

Check if MySQL is configured to run with `--gtid-mode=ON` using the following command in the MySQL CLI:

```
show variables like '%gtid%';
```

Remember the value of the gtid-mode variable because it affects the steps in the section Upgrade MySQL and the Cisco Crosswork Situation Manager Schema.

Official instructions for upgrading from v5.6.26 to v5.7.22 are here: https://dev.mysql.com/doc/refman/5.7/en/upgrading.html and the steps are summarized below. It is recommended that you perform a backup of the database before continuing.

Stop the MySQL service and change the password as appropriate:

```
mysql -u root --password='' --execute="SET GLOBAL innodb_fast_shutdown=0"
service mysqld stop
```

Upgrade Cisco Crosswork Situation Manager

Please note that there are some new moogsoft packages (moogsoft-integrations, moogsoft-integrations-ui) which are required by Cisco Crosswork Situation Manager v7.0.0. As a result, you must perform an install of those packages and an upgrade of the other packages at the same time.

This is performed using a 'yum shell' script.

If you are performing an offline upgrade, you must download the new MySQL packages manually before continuing, as specified in Pre-Install Cisco Crosswork Situation Manager - Offline RPMs.

Choose the upgrade that corresponds to your installation:

- Single host from a remote repository. Run the following commands to perform the upgrade:

```
echo 'upgrade moogsoft-common-7.0.0' > upgrade_to_700.sh
echo 'upgrade moogsoft-db-7.0.0' >> upgrade_to_700.sh
echo 'install moogsoft-integrations-7.0.0' >> upgrade_to_700.sh
echo 'install moogsoft-integrations-ui-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-mooms-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-search-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-server-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-ui-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-utils-7.0.0' >> upgrade_to_700.sh
echo 'remove moogsoft-lams' >> upgrade_to_700.sh
echo 'run' >> upgrade_to_700.sh
cat upgrade_to_700.sh | yum shell -y
```

  If this is an offline upgrade and the MySQL RPMs are in the current directory, you should the following script instead:

```
echo 'upgrade moogsoft-common-7.0.0' > upgrade_to_700.sh
echo 'upgrade moogsoft-db-7.0.0' >> upgrade_to_700.sh
echo 'install moogsoft-integrations-7.0.0' >> upgrade_to_700.sh
echo 'install moogsoft-integrations-ui-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-mooms-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-search-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-server-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-ui-7.0.0' >> upgrade_to_700.sh
echo 'upgrade moogsoft-utils-7.0.0' >> upgrade_to_700.sh
echo 'remove moogsoft-lams' >> upgrade_to_700.sh
echo 'upgrade mysql-community-libs-compat-5.7.22-1.el7.x86_64.rpm' >>
upgrade_to_700.sh
echo 'upgrade mysql-community-server-5.7.22-1.el7.x86_64.rpm' >>
upgrade_to_700.sh
echo 'upgrade mysql-community-client-5.7.22-1.el7.x86_64.rpm' >>
upgrade_to_700.sh
echo 'upgrade mysql-community-common-5.7.22-1.el7.x86_64.rpm' >>
upgrade_to_700.sh
echo 'upgrade mysql-community-libs-5.7.22-1.el7.x86_64.rpm' >>
upgrade_to_700.sh
echo 'run' >> upgrade_to_700.sh
cat upgrade_to_700.sh | yum shell -y
```

- Single host from same machine where the RPMs have been downloaded locally and are in the current folder. Run the following commands to perform the upgrade:

```
echo 'upgrade moogsoft-common-7.0.0-116.x86_64.rpm' > upgrade_to_700.
sh
echo 'install moogsoft-integrations-7.0.0-19744.x86_64.rpm' >>
upgrade_to_700.sh
echo 'install moogsoft-integrations-ui-7.0.0-19744.x86_64.rpm' >>
upgrade_to_700.sh
echo 'upgrade moogsoft-db-7.0.0-116.x86_64.rpm' >> upgrade_to_700.sh
echo 'upgrade moogsoft-mooms-7.0.0-116.x86_64.rpm' >> upgrade_to_700.
sh
echo 'upgrade moogsoft-search-7.0.0-116.x86_64.rpm' >> upgrade_to_700.
sh
echo 'upgrade moogsoft-server-7.0.0-116.x86_64.rpm' >> upgrade_to_700.
sh
echo 'upgrade moogsoft-ui-7.0.0-116.x86_64.rpm' >> upgrade_to_700.sh
echo 'upgrade moogsoft-utils-7.0.0-116.x86_64.rpm' >> upgrade_to_700.
sh
echo 'remove moogsoft-lams' >> upgrade_to_700.sh
echo 'run' >> upgrade_to_700.sh
cat upgrade_to_700.sh | yum shell -y
```

- Distributed installs. Select the relevant update commands to run depending on which package(s) are installed on the current host. For example:

    1. HOST1 running common (required), moogfarmd server, Cisco Crosswork Situation Manager UI, and Cisco Crosswork Situation Manager Integrations UIpackages:

    ```
    echo 'upgrade moogsoft-common-7.0.0' > upgrade_to_700.sh
    echo 'upgrade moogsoft-server-7.0.0' >> upgrade_to_700.sh
    echo 'upgrade moogsoft-ui-7.0.0' >> upgrade_to_700.sh
    echo 'install moogsoft-integrations-ui-7.0.0' >> upgrade_to_700.
    sh
    echo 'run' >> upgrade_to_700.sh
    cat upgrade_to_700.sh | yum shell -y
    ```

    2. HOST2 running common (required), database, and search packages:

    ```
    echo 'upgrade moogsoft-common-7.0.0' > upgrade_to_700.sh
    echo 'upgrade moogsoft-db-7.0.0' >> upgrade_to_700.sh
    echo 'upgrade moogsoft-search-7.0.0' >> upgrade_to_700.sh
    echo 'run' >> upgrade_to_700.sh
    cat upgrade_to_700.sh | yum shell -y
    ```

    3. HOST3 running common (required), Integrations, message bus, and utilities packages:

    ```
    echo 'upgrade moogsoft-common-7.0.0' > upgrade_to_700.sh
    echo 'install moogsoft-integrations-7.0.0' >> upgrade_to_700.sh
    echo 'remove moogsoft-lams' >> upgrade_to_700.sh
    echo 'upgrade moogsoft-mooms-7.0.0' >> upgrade_to_700.sh
    echo 'upgrade moogsoft-utils-7.0.0' >> upgrade_to_700.sh
    echo 'run' >> upgrade_to_700.sh
    cat upgrade_to_700.sh | yum shell -y
    ```

Merge the Latest Config File Changes

Manually merge and compare `.rpmsave` versions of files with the new versions of those files. Add any new properties to the older versions of the files.

To find files which have been changed/moved/deleted run these commands:

```
find $MOOGSOFT_HOME -name '*.rpmsave'
find /etc/init.d/ -name '*.rpmsave'
```

An example command to see what differences are present in the `$MOOGSOFT_HOME/config/system.conf` file is shown below:

```
diff -u $MOOGSOFT_HOME/config/system.conf $MOOGSOFT_HOME/config/system.
conf.rpmsave
```

An example of the process to merge the differences is shown below:

1. Rename the new versions of the files without the `.rpmsave` extension to end with `.bak`.
2. Merge the `.rpmsave` file with the new `.bak` file by adding new properties/config where needed so the structure matches the new version of the file.
3. Rename the `.rpmsave` file to delete the `.rpmsave` extension.

Optional: Decrease log verbosity of Elasticsearch

Run the following command to change the default log level for Elasticsearch. By default it is set to `info` but you can change it to `warn` to reduce the size of the Elasticsearch logs.

```
sed -i -e "s;rootLogger.level = info;rootLogger.level = warn;g" /etc
/elasticsearch/log4j2.properties
```

Restart Elasticsearch, RabbitMQ, and Nginx

Run the following commands to restart Elasticsearch, RabbitMQ, and Nginx now that they have been upgraded:

```
service elasticsearch restart
service rabbitmq-server restart
service nginx restart
```

Upgrade MySQL and the Cisco Crosswork Situation Manager Schema

To upgrade MySQL and the Cisco Crosswork Situation Manager Schema:

1. Update `/etc/my.cnf` with two new properties using the script below:

   ```
   sed -i 's/\(table-open-cache.*\)/\1\nmax_prepared_stmt_count =
   1048576\ngroup_concat_max_len = 1048576/' /etc/my.cnf
   ```

2. Restart MySQL:

   ```
   service mysqld restart
   ```

3 If the `gtid-mode` is set to **OFF**, based on the command run earlier in the upgrade, run the MySQL upgrade utility. Provide the MySQL root password when prompted or just press <return> if no password set. Then restart MySQL to ensure any changes to system tables are saved:

```
mysql_upgrade -u root -p
service mysqld restart
```

More information on GTID and mysql_upgrade is here: https://dev.mysql.com/doc/refman/5.7/en/replication-gtids-restrictions.html#replication-gtids-restrictions-mysql_upgrade.

4 To upgrade the Cisco Crosswork Situation Manager database, you need to provide the `moog_db_auto_upgrader` utility with the credentials of a *database* user with super privileges. For single-host installs where MySQL was installed as part of the Cisco Crosswork Situation Manager deployment, you can use the default 'root' user.

Run the following command after substituting the `<MySQL-SuperUsername>` argument:

```
bash $MOOGSOFT_HOME/bin/utils/moog_db_auto_upgrader -t 7.0.0 -u
<MySQL-SuperUsername>
```

Enter the password for that user. You can provide the password to the utility with the `-p` flag but this is not recommended in non-test deployments for security reasons.

5 If you see an error such as this:

```
WARN : [main][20180710 20:40:11.376 +0000] [CAutoUpgrader.java:446]
+|Encountered errors during execution, exiting...|+

The 643_moogdb_upgrade_all.sql upgrade script contains the following
at line 70:
```

Change the file `$MOOGSOFT_HOME/etc/moog/schema_upgrades/6.4.2-6.4.3/643_moogdb_upgrade_all.sql` to remove these lines:

```
UPDATE system_config
SET properties = JSON_SET(properties, "$.\"seed_alert_filter\"", "{}")
WHERE id=4;

UPDATE system_config
SET properties = JSON_SET(properties, "$.\"seed_alert_filter\"", "{}")
WHERE id=5;

UPDATE system_config
SET properties = JSON_SET(properties, "$.\"seed_alert_filter\"", "{}")
WHERE id=6;
```

And replace them with:

```
UPDATE system_config
    SET properties = JSON_SET(properties, "$.\"seed_alert_filter\"",
"{}")
    WHERE config_type = 'Recipe' AND  properties ->> '$.
seed_alert_filter' IS NULL;
```

Then re-run the moog_db_auto_upgrader command.

Upgrade the Apache Tomcat Webapps and Configuration Files

The Apache Tomcat service script has been updated in this version and the new script needs to be used.

If any customizations have been made to the `/etc/init.d/apache-tomcat` script, e.g. an `-Xmx` change, please make a backup before continuing.

Replace the old service script with the new one using the following command:

```
cp -f $MOOGSOFT_HOME/etc/cots/tomcat/apache-tomcat.sh /etc/init.d/apache-
tomcat
```

If any customizations were made in the old script, they should be made to the new version of the script before continuing.

Similarly, the `$APPSERVER_HOME/conf/server.xml` and `$APPSERVER_HOME/conf/context.xml` files need to be replaced with new versions. The command below deploys the new ones. Back up the existing ones if customizations have been made to them.

Now, perform the upgrade of the webapps and `server.xml/context.xml`.

To upgrade using the script provided, run the following command. This is recommended for standard or single-host installs.

```
$MOOGSOFT_HOME/bin/utils/moog_init_ui.sh -wf
```

If Apache Tomcat startup or shutdown failures are reported when running the above command, such as "Web apps are not rebuilding, please check catalina.out", then you should forcibly shut down Apache Tomcat using `kill -9` on the Apache Tomcat java PID, and then repeat the step above.

Alternatively, to upgrade manually, follow these steps:

1. Stop Apache Tomcat:

   ```
   service apache-tomcat stop
   ```

2. Check the Apache Tomcat java process is no longer running:

   ```
   service apache-tomcat status
   ```

3. Delete the existing extracted webapps:

   ```
   rm -rf $APPSERVER_HOME/webapps/{moogpoller,moogsvr,toolrunner,graze,
   events,situation_similarity}
   ```

4. Copy in the new webapp WAR files and replace the old ones:

   ```
   cp -f $MOOGSOFT_HOME/lib/{moogpoller,moogsvr,toolrunner,graze,events,
   situation_similarity}.war $APPSERVER_HOME/webapps/
   ```

5. Restart Apache Tomcat:

```
service apache-tomcat start
```

Update the Integrations

To update the Integrations:

1. Run the following command needs to be run to extract Integrations:

```
bash $MOOGSOFT_HOME/bin/utils/integration_installer -a -l WARN
```

2. After you finish the upgrade, un-install and re-install any of the following Ticketing Integrations that were present before this upgrade:

   - HP OMi
   - HP NNMi
   - Zenoss
   - Solarwinds
   - JDBC
   - Zabbix
   - Dynatrace APM
   - vSphere
   - vCenter
   - vRealize Log Insight
   - Email

Confirm All Services are Running and Start moogfarmd

To confirm all services are running:

1. Run the following commands to check all required services are running and to start moogfarmd:

```
service apache-tomcat status
service moogfarmd start
service nginx status
service elasticsearch status
```

2. Run the following command to re-enable the `events_analyser` cronjobs:

```
(crontab -l | sed -e 's/^\#\+\(.*events_analyser.*\)/\1/') | crontab -
```

3. Run the following command to restart any previously running Integrations:

```
service moogstartupd restart
```

Re-index Elasticsearch

Run the following command on the moogsoft-search/Elasticsearch server to remove the old Elasticsearch indexes:

```
curl -XDELETE 'http://localhost:9200/alerts/' && curl -XDELETE
'http://localhost:9200/situations/'
```

If the command completes successfully, the following message should be shown on the screen:

```
```

```
{"acknowledged":true}{"acknowledged":true}
```

Run the following command to re-index the alerts and Situations in moogfarmd. This requires moogfarmd to be running.

```
$MOOGSOFT_HOME/bin/utils/moog_indexer --now
```

The re-index occurs in the background and may take a while to complete. This depends on the number of alerts and Situations in your system.

Migrate the Java Keystore

The upgrade includes a new version of the Java Runtime, so you need to migrate any certificates stored in the old Java keystore to the new `JAVA_HOME` keystore.

If you did not manually add certificates to the old store, you can skip this step.

Verify the Upgrade

You can verify the upgrade manually or using automatic utilities.

Verify the Upgrade Manually

Perform the following basic steps to ensure the upgrade to Cisco Crosswork Situation Manager v7.0.0 was successful:

1. Check that the UI login page displays "Version 7 .0.0" at the top.
2. Check that the UI "Support Information" window correctly indicates the current version as "7.0.0" and shows the correct schema upgrade history.

Verify the Upgrade Using Automatic Utilities

Run the following automatic utilities to ensure that the upgrade to Cisco Crosswork Situation Manager v7.0.0 was successful:

1 Confirm all Moog files have been deployed correctly within `$MOOGSOFT_HOME` using this utility:

```
$MOOGSOFT_HOME/bin/utils/moog_install_validator.sh
```

2 Confirm all Apache Tomcat files have been deployed correctly within `$MOOGSOFT_HOME` using this utility:

```
$MOOGSOFT_HOME/bin/utils/tomcat_install_validator.sh
```

3 Confirm the schema has been upgraded successfully using this utility:

```
$MOOGSOFT_HOME/bin/utils/moog_db_validator.sh
```

4 Confirm that all the steps are successful. If there are some webapp differences then this can be resolved using moog_init_ui.sh -w which will extract the webapps with the right files.

5 Some schema differences may be valid (e.g. `custom_info` related). If there are more substantial differences, you should investigate further to verify that all the pre-requisite upgrade scripts have been applied in the right order:

```
mysql -u root <moogdb_database_name> -e "select * from
schema_upgrades;"
mysql -u root <moog_reference_database_name> -e "select * from
schema_upgrades;"
```

Troubleshooting

If you have any issues, refer to Troubleshooting.

Upgrade v6.5.0 or v6.5.0.x to v7.0.0

This topic describes the upgrade procedure of RPM deployments from Cisco Crosswork Situation Manager v6.5.0 or v6.5.0.x to Cisco Crosswork Situation Manager v7.0.0. Refer to Upgr ade Cisco Crosswork Situation Manager - RPM Deployments for general information and links to upgrades for other versions.

The following steps outline this workflow:

Before You Begin

Red Hat Enterprise Linux 6/CentOS 6 (RHEL6) is not supported for Cisco Crosswork Situation Manager v7.0.0. See Deprecation Notice: RHEL 6. If your previous installation runs on RHEL6, see the Upgrade - Migrate from RHEL6 to RHEL7 guide before carrying out this migration.

To prepare for an offline upgrade, where the Cisco Crosswork Situation Manager packages reside in a 'local' yum repository, follow steps 1-8 in section "Prepare the Local yum Repositories" in the Pre-Install Cisco Crosswork Situation Manager - Offline RPMs guide. Then follow this guide to perform the upgrade. Take into account any step-specific notes regarding Offline upgrades.

Back Up the Existing System

To back up the existing system:

1. Back up `$MOOGSOFT_HOME`.
2. Take a snapshot (for VMs).
3. Back up MySQL.

Stop Services and Processes

To stop the services and processes:

1. Stop `moogfarmd`:

   ```
   service moogfarmd stop
   ```

   Alternatively, if the moogfarmd instance was started via the UI, you should stop it via the UI at this point.
2. Ensure no more farmd processes are running with this command:

   ```
   kill -9 $(ps -ef | grep java | grep farm | awk '{print $2}') 2>/dev
   /null
   ```

3. Stop Apache Tomcat:

   ```
   ```

```
service apache-tomcat stop
```

4. Stop the LAMs. You can do this via their service scripts:

```
service <lam_service_name> stop
```

   Alternatively, stop their running processes using the command:

```
kill <pid>
```

5. Ensure all the above processes have been stopped before continuing:

```
service <service_name> status
```

6. Disable the `events_analyser` from running during the upgrade process:

   1. Run the following command to comment out the relevant lines in `crontab`:

```
(crontab -l | sed -e 's/^\(.*events_analyser.*\)$/#\1/') |
crontab -
```

   2. Run the following command to stop any running `events_analyser` processes:

```
ps -ef | grep java | egrep 'events_analyser' | awk '{print $2}'
| xargs kill 2>/dev/null
```

Upgrade Cisco Crosswork Situation Manager

Choose the upgrade command that corresponds to your installation:

- Single host from a remote or local yum repository. Run the following commands to perform the upgrade:

```
yum -y upgrade moogsoft-integrations-7.0.0 moogsoft-integrations-ui-
7.0.0 moogsoft-common-7.0.0 moogsoft-server-7.0.0 moogsoft-ui-7.0.0
moogsoft-db-7.0.0 moogsoft-search-7.0.0 moogsoft-mooms-7.0.0 moogsoft-
utils-7.0.0
```

- Single host from the same machine where the RPMs have been downloaded locally and are in the current folder. Run the following commands to perform the upgrade:

```
yum -y upgrade moogsoft-*7.0.0*.rpm
```

- Distributed installs. Select the relevant update commands to run depending on which package(s) are installed on the current host. For example:

   1. HOST1 running Integrations, Integrations UI, common (required), moogfarmd server, and Cisco Crosswork Situation Manager UI packages:

```
yum -y upgrade moogsoft-integrations-7.0.0 moogsoft-integrations-
ui-7.0.0 moogsoft-common-7.0.0 moogsoft-server-7.0.0 moogsoft-ui-
7.0.0
```

2. HOST2 running common (required), utilities, database, and search packages:

```
yum -y upgrade moogsoft-common-7.0.0 moogsoft-utils-7.0.0
moogsoft-db-7.0.0 moogsoft-search-7.0.0
```

3. HOST3 running common (required), message bus, and utilities packages:

```
yum -y upgrade moogsoft-common-7.0.0 moogsoft-mooms-7.0.0
moogsoft-utils-7.0.0
```

Merge the Latest Config File Changes

Manually merge and compare `.rpmsave` versions of files with the new versions of those files. Add any new properties to the older versions of the files.

To find files which have been changed, moved, or deleted, run these commands:

```
find $MOOGSOFT_HOME -name '*.rpmsave'
find /etc/init.d/ -name '*.rpmsave'
```

An example command to see what differences are present in the `$MOOGSOFT_HOME/config/system.conf` file is shown below:

```
diff -u $MOOGSOFT_HOME/config/system.conf $MOOGSOFT_HOME/config/system.
conf.rpmsave
```

An example of the process to merge the differences is shown below:

1 Rename the new versions of the files, without the `.rpmsave` extension, to end with `.bak`.
2 Merge the `.rpmsave` file with the new `.bak` file by adding new properties/config where needed so the structure matches the new version of the file.
3 Rename the `.rpmsave` file to delete the `.rpmsave` extension.

Optional: Decrease Log Verbosity of Elasticsearch

To decrease the log verbosity of Elasticsearch:

1 Run the following command to change the default log level for Elasticsearch. By default it is set to `info` but you can change it to `warn` to reduce the size of the Elasticsearch logs.

```
sed -i -e "s;rootLogger.level = info;rootLogger.level = warn;g" /etc
/elasticsearch/log4j2.properties
```

2 Restart the Elasticsearch service:

```
service elasticsearch restart
```

Upgrade the Cisco Crosswork Situation Manager Schema

To upgrade the Cisco Crosswork Situation Manager database, you need to provide the `moog_db_auto_upgrader` utility with the credentials of a *database* user with super privileges. For single-host installs where MySQL was installed as part of the Cisco Crosswork Situation Manager deployment, you can use the default 'root' user.

1 Run the following command after substituting the `<MySQL-SuperUsername>` argument:

```
bash $MOOGSOFT_HOME/bin/utils/moog_db_auto_upgrader -t 7.0.0 -u
<MySQL-SuperUsername>
```

2 Enter the password for that user. You can provide the password to the utility with the `-p` flag but this is not recommended in non-test deployments for security reasons.

Upgrade the Apache Tomcat Webapps and Configuration Files

The `$APPSERVER_HOME/conf/server.xml` and `$APPSERVER_HOME/conf/context.xml` files have been changed in this release. If the existing deployment has customized versions of those files, please make a backup before continuing.

To upgrade using the script provided, run the following command. This is recommended for standard or single-host installs.

```
$MOOGSOFT_HOME/bin/utils/moog_init_ui.sh -wf
```

If Apache Tomcat startup or shutdown failures are reported when running the above command, such as "Web apps are not rebuilding, please check catalina.out", then you should forcibly shut down Apache Tomcat using `kill -9` on the Apache Tomcat java PID, and then repeat the step above.

Alternatively, to upgrade manually, follow these steps:

1. Stop Apache Tomcat:

```
service apache-tomcat stop
```

2. Check the Apache Tomcat java process is no longer running:

```
service apache-tomcat status
```

3. Delete the existing extracted webapps:

```
rm -rf $APPSERVER_HOME/webapps/{moogpoller,moogsvr,toolrunner,graze,
events,situation_similarity}
```

4. Copy in the new webapp WAR files and replace the old ones:

```
cp -f $MOOGSOFT_HOME/lib/{moogpoller,moogsvr,toolrunner,graze,events,
situation_similarity}.war $APPSERVER_HOME/webapps/
```

5. Restart Apache Tomcat:

```
service apache-tomcat start
```

If customizations had been made to `server.xml` or `context.xml`, update the new version of those files and restart the Apache Tomcat service.

Update the Integrations

To update the Integrations:

1. Run the following command to extract the Integrations:

```
bash $MOOGSOFT_HOME/bin/utils/integration_installer -a -l WARN
```

2. After you finish the upgrade, un-install and re-install any of the following Ticketing Integrations that were present before this upgrade:

   - HP OMi
   - HP NNMi
   - Zenoss
   - Solarwinds
   - JDBC
   - Zabbix
   - Dynatrace APM
   - vSphere
   - vCenter
   - vRealize Log Insight
   - Email

Confirm All Services are Running and Start moogfarmd

To confirm all services are running:

1. Run the following commands to check all required services are running and to start moogfarmd:

```
service apache-tomcat status
service moogfarmd start
service nginx status
service elasticsearch status
```

2. Run the following command to re-enable the `events_analyser` cronjobs:

```
(crontab -l | sed -e 's/^\#\+\(.*events_analyser.*\)/\1/') | crontab -
```

3. Run the following command to restart any previously running Integrations:

```
service moogstartupd restart
```

Migrate the Java Keystore

The upgrade includes a new version of the Java Runtime, so you need to migrate any certificates stored in the old Java keystore to the new `JAVA_HOME` keystore.

If you did not manually add certificates to the old store, you can skip this step.

Verify the Upgrade

You can verify the upgrade manually or using automatic utilities.

Verify the Upgrade Manually

Perform the following basic steps to ensure that the upgrade to Cisco Crosswork Situation Manager v7.0.0 was successful:

1. Check that the UI login page displays "Version 7 .0.0" at the top.
2. Check that the UI "Support Information" window correctly indicates the current version as "7.0.0" and shows the correct schema upgrade history.

Verify the Upgrade Using Automatic Utilities

Run the following automatic utilities to ensure that the upgrade to Cisco Crosswork Situation Manager v7.0.0 was successful:

1 Confirm all Moog files have been deployed correctly within `$MOOGSOFT_HOME` using this utility:

```
$MOOGSOFT_HOME/bin/utils/moog_install_validator.sh
```

2 Confirm all Apache Tomcat files have been deployed correctly within `$MOOGSOFT_HOME` using this utility:

```
$MOOGSOFT_HOME/bin/utils/tomcat_install_validator.sh
```

3 Confirm the schema has been upgraded successfully using this utility:

```
$MOOGSOFT_HOME/bin/utils/moog_db_validator.sh
```

4 Confirm that all the steps are successful. If there are some webapp differences, you can resolve them using `moog_init_ui.sh -w` which extracts the webapps with the right files.
5 Some schema differences may be valid (e.g. `custom_info` related). If there are more substantial differences, you should investigate further to verify that all the pre-requisite upgrade scripts have been applied in the right order:

```
mysql -u root <moogdb_database_name> -e "select * from
schema_upgrades;"
mysql -u root <moog_reference_database_name> -e "select * from
schema_upgrades;"
```

Troubleshooting

If you have any issues, refer to Troubleshooting.

Upgrade - Non-root Deployments

This guide provides instruction on how to upgrade to Cisco Crosswork Situation Manager ESR 7.0.0 from Cisco

Crosswork Situation Manager v6.5.0. Please see Upgrade Non-root v6.5.0 to Non-root v7.0.0 for detailed upgrade

procedures.

Upgrade Non-root v6.5.0 to Non-root v7.0.0

This topic describes the upgrade procedure of non-root deployments from Cisco Crosswork Situation Manager v6.5.0 non-root to Cisco

Crosswork Situation Manager v7.0.0 non-root. The following steps outline this workflow:

- Before You Begin
- Back Up the Existing System
- Stop Processes
- Upgrade Cisco Crosswork Situation Manager
- Migrate Data Directories and Restart MySQL and Elasticsearch
- Merge the Latest Config File Changes
- Upgrade the Cisco Crosswork Situation
- Manager Schema Upgrade Apache
  Tomcat and Nginx

Before You Begin

Red Hat Enterprise Linux 6/CentOS 6 (RHEL6) is not supported for Cisco Crosswork Situation Manager v7.0.0. See Deprecation Notice: RHEL 6

To do this workflow, you must be logged in as the same non-root user that installed and runs the existing Cisco Crosswork Situation Manager v6.5.0 software. Download the New Installer

Run the following command to download the installer:

```
curl -L -O "https://<username>:<password>@speedy.moogsoft.com/installer
/moogsoft-aiops-7.0.0.tgz"
```

Back Up the Existing System

To back up the existing system:

1. Back up `$MOOGSOFT_HOME`.
2. Take a snapshot (for VMs).
3. Back up MySQL.

Stop Processes

1. Stop `moog_farmd`:

    ```
    $MOOGSOFT_HOME/bin/utils/process_cntl moog_farmd stop
    ```

2. Stop Apache Tomcat:

    ```
    $MOOGSOFT_HOME/bin/utils/process_cntl apache-tomcat stop
    ```

3. Stop the LAMs. You can use either `process_cntl` or `kill` .
   Using `process_cntl`:

    ```
    $MOOGSOFT_HOME/bin/utils/process_cntl <lam_name> stop
    ```

   Using `kill`:

    ```
    kill -9 $(ps -ef | grep java | grep lam | awk '{print $2}') 2>/dev
    /null
    ```

4. Disable the `events_analyser` from running during the upgrade process:
    1. Run the following command to comment out the relevant lines in `crontab`:

        ```
        (crontab -l | sed -e 's/^\(.*events_analyser.*\)$/#\1/') |
        crontab -
        ```

2. Run the following command to stop any running `events_analyser` processes:

```
ps -ef | grep java | egrep 'events_analyser|moog_indexer' | awk
'{print $2}' | xargs kill 2>/dev/null
```

5. Stop `mysql`, `rabbitmq`, `elasticsearch`, `nginx`, and `tomcat`:

```
$MOOGSOFT_HOME/bin/utils/process_cntl mysqld stop
$MOOGSOFT_HOME/bin/utils/process_cntl elasticsearch stop
$MOOGSOFT_HOME/bin/utils/process_cntl rabbitmq stop
$MOOGSOFT_HOME/bin/utils/process_cntl nginx stop
```

Upgrade Cisco Crosswork Situation Manager

Run the following commands to perform the upgrade:

```
tar -xf moogsoft-aiops-7.0.0.tgz
bash moogsoft-aiops-install-7.0.0.sh
```

Follow the instructions. It detects the v6.5.0 installation and performs an upgrade (a side-by-side deployment).

Migrate Data Directories and Restart MySQL and Elasticsearch

Run the following commands to migrate the data directories for MySQL, RabbitMQ, and Elasticsearch into the new locations:

```
mkdir -p $MOOGSOFT_HOME/var/lib/{elasticsearch,mysql,rabbitmq}
cp -frp $MOOGSOFT_HOME/dist/6.5.0*/cots/elasticsearch/data $MOOGSOFT_HOME
/var/lib/elasticsearch/
sed -i "s;#path.data: /path/to/data;path.data: $MOOGSOFT_HOME/var/lib
/elasticsearch/data;g" $MOOGSOFT_HOME/cots/elasticsearch/config
/elasticsearch.yml
cp -frp $MOOGSOFT_HOME/dist/6.5.0*/cots/mysql/* $MOOGSOFT_HOME/var/lib
/mysql/
sed -i "s;$MOOGSOFT_HOME/cots/mysql/;$MOOGSOFT_HOME/var/lib/mysql/;" ~/.my.
cnf
cp -frp $MOOGSOFT_HOME/dist/6.5.0*/cots/rabbitmq-server/var/lib/rabbitmq
/mnesia $MOOGSOFT_HOME/var/lib/rabbitmq/
```

Run the following command to change the default log level for Elasticsearch. By default it is set to `info` but you can change it to `warn` to reduce the size of the Elasticsearch logs.

```
sed -i -e "s;rootLogger.level = info;rootLogger.level = warn;g"
$MOOGSOFT_HOME/cots/elasticsearch/config/log4j2.properties
```

Run the following commands to restart the processes:

```
$MOOGSOFT_HOME/bin/utils/process_cntl mysqld start
$MOOGSOFT_HOME/bin/utils/process_cntl elasticsearch start
$MOOGSOFT_HOME/bin/utils/process_cntl rabbitmq start
```

Merge the Latest Config File Changes

The top-level `$MOOGSOFT_HOME/config` and `$MOOGSOFT_HOME/bots` folders are the master folder locations for config and bot files. The new 'default' v7.0.0 versions of these files are stored under `$MOOGSOFT_HOME/dist/7.0.0/config/` and `$MOOGSOFT_HOME/dist/7.0.0/bots/` respectively.

1. Identify any new "`config`" files added in v7.0.0. For example:

```
diff -rq $MOOGSOFT_HOME/config $MOOGSOFT_HOME/dist/7.0.0/config/ |
grep -i 'Only'
```

2. Copy the new 'config' files into the `$MOOGSOFT_HOME/config` folder.
3. Identify any new 'contrib' files added in v7.0.0. For example:

```
diff -rq $MOOGSOFT_HOME/contrib $MOOGSOFT_HOME/dist/7.0.0/contrib/ |
grep -i 'Only'
```

4. Copy the new 'contrib' files into the `$MOOGSOFT_HOME/ contrib` folder.
5. Identify any new 'bot' files added in v7.0.0. For example:

```
diff -rq $MOOGSOFT_HOME/bots $MOOGSOFT_HOME/dist/7.0.0/bots/ | grep -
i 'Only'
```

6. Copy the new 'bot' files into the `$MOOGSOFT_HOME/bots` folder (including the subdirectory if appropriate).
7. Identify the 'config' files that have changed between v6.5.0 and v7.0.0. For example:

```
diff -rq $MOOGSOFT_HOME/dist/6.5.0*/config $MOOGSOFT_HOME/dist/7.0.0
/config | grep -i 'differ'
```

8. Update the files in `$MOOGSOFT_HOME/config/` with any changes introduced in the v7.0.0 versions of these files.
9. Identify the 'contrib' files that have changed between v6.5.0 and v7.0.0. For example:

```
diff -rq $MOOGSOFT_HOME/dist/6.5.0*/contrib $MOOGSOFT_HOME/dist/7.0.0
/contrib | grep -i 'differ'
```

10. Update the files in `$MOOGSOFT_HOME/contrib/` with any changes introduced in the v7.0.0 versions of these files.
11. Identify the 'bot' files that have changed between v6.5.0 and v7.0.0. For example:

```
diff -rq $MOOGSOFT_HOME/dist/6.5.0*/bots $MOOGSOFT_HOME/dist/7.0.0
/bots | grep -i 'differ'
```

12. Update the files in $MOOGSOFT_HOME/bots/ with any changes introduced in the v7.0.0 versions of these files.

Upgrade the Cisco Crosswork Situation Manager Schema

To upgrade the Cisco Crosswork Situation Manager database, you must provide the `moog_db_auto_upgrader` utility with the credentials of a *database* user with super privileges. For single-host installs where MySQL was installed as part of the Cisco Crosswork Situation Manager deployment, you can use the default 'root' user.

1. Run the following command after substituting the `<MySQL-SuperUsername>` argument:

```
bash $MOOGSOFT_HOME/bin/utils/moog_db_auto_upgrader -t 7.0.0 -u
<MySQL-SuperUsername>
```

2. Enter the password for that user. You can provide the password to the utility with the `-p` flag but this is not recommended in non-test deployments for security reasons.

Upgrade Apache Tomcat and Nginx

Enter the following commands to upgrade Apache Tomcat and Nginx:

```
$MOOGSOFT_HOME/bin/utils/moog_init_ui.sh -qtnfwz $($MOOGSOFT_HOME/bin/utils
/moog_config_reader -k mooms.zone) --accept-eula
```

Migrate the certificates for the previous deployment of Nginx to the new deployment. Use the following commands as an example of how to do this in the general case where SSL terminates in Nginx (default configuration):

```
CERT_PATH_PEM=$(grep 'ssl_certificate ' $MOOGSOFT_HOME/dist/6.5.0*/cots
/nginx/config/conf.d/moog-ssl.conf)
CERT_PATH_KEY=$(grep 'ssl_certificate_key' $MOOGSOFT_HOME/dist/6.5.0*/cots
/nginx/config/conf.d/moog-ssl.conf)
cp -f $(echo "${CERT_PATH_PEM:0:-1}" | awk '{print $2}') $MOOGSOFT_HOME
/dist/7.0.0/cots/nginx/ssl/
cp -f $(echo "${CERT_PATH_KEY:0:-1}" | awk '{print $2}') $MOOGSOFT_HOME
/dist/7.0.0/cots/nginx/ssl/
sed -i "s|.*ssl_certificate .*|${CERT_PATH_PEM}|" $MOOGSOFT_HOME/dist/7.0.0
/cots/nginx/config/conf.d/moog-ssl.conf
sed -i "s|.*ssl_certificate_key.*|${CERT_PATH_KEY}|" $MOOGSOFT_HOME/dist/7.
0.0/cots/nginx/config/conf.d/moog-ssl.conf
```

Restart Nginx

```
$MOOGSOFT_HOME/bin/utils/process_cntl nginx restart
```

Update the Integrations

To update the integrations:

1. Run the following command to extract the Integrations:

```
bash $MOOGSOFT_HOME/bin/utils/integration_installer -a -l WARN;
for file in $(diff -qr $MOOGSOFT_HOME/dist/7.0.0/ui/integrations/ \
$MOOGSOFT_HOME/dist/6.5.0*/ui/integrations/ | egrep -vi 'only in.*7.
0.0|differ' \
```

```
| awk '{print $NF}'); do cp -rp "$MOOGSOFT_HOME/dist/6.5.0*/ui
/integrations/$file" \
$MOOGSOFT_HOME/dist/7.0.0/ui/integrations/; done
```

2. After you finish the upgrade, uninstall and reinstall any of the following Ticketing Integrations that were present before the install:

   - HP OMi
   - HP NNMi
   - Zenoss
   - Solarwinds
   - JDBC
   - Zabbix
   - Dynatrace APM
   - vSphere
   - vCenter
   - vRealize Log Insight
   - Email

Confirm All Processes are Running and Start moogfarmd

To confirm all services are running:

1. Run the following commands to verify that all required processes are running and to start moogfarmd:

```
$MOOGSOFT_HOME/bin/utils/process_cntl apache-tomcat status
$MOOGSOFT_HOME/bin/utils/process_cntl moog_farmd start
$MOOGSOFT_HOME/bin/utils/process_cntl nginx status
$MOOGSOFT_HOME/bin/utils/process_cntl elasticsearch status
```

2. Run the following command to re-enable the `events_analyser` cronjobs:

```
(crontab -l | sed -e 's/^\#\+\(.*events_analyser.*\)/\1/') | crontab -
```

3. Run the following command to restart any previously running UI-based Integrations:

```
bash $MOOGSOFT_HOME/bin/utils/startup_cntl;
```

Migrate the Java Keystore

The upgrade includes a new version of the Java Runtime, so you need to migrate any certificates stored in the old Java keystore into the new keystore, which is now the JRE under `$MOOGSOFT_HOME/cots/jre/`.

The previous JRE is under `$MOOGSOFT_HOME/dist/6.5.0/cots/jre`.

If you did not manually add certificates to the old store, you can skip this step.

Verify the Upgrade

You can verify the upgrade manually or using automatic utilities.

Verify the Upgrade Manually

Perform the following basic steps to ensure the upgrade to Cisco Crosswork Situation Manager v7.0.0 was successful:

1. Verify that the UI login page displays "Version 7.0.0" at the top.
2. Check that the UI "Support Information" window correctly indicates the current version as "7.0.0" and shows the correct schema upgrade history.

Verify the Upgrade Using Automatic Utilities

Run the following automatic utilities to ensure that the upgrade to 7.0.0 was successful:

1. Verify that Moog files are deployed correctly within `$MOOGSOFT_HOME` using this utility:

```
$MOOGSOFT_HOME/bin/utils/moog_install_validator.sh
```

2. Verify that all Apache Tomcat files are deployed correctly within `$MOOGSOFT_HOME` using this utility:

```
$MOOGSOFT_HOME/bin/utils/tomcat_install_validator.sh
```

3. Verify that all the steps are successful. If there are some webapp differences, run the following command to extract the webapp with the correct files:

```
$MOOGSOFT_HOME/bin/utils/moog_init_ui.sh -w
```

4. Verify that the schema has upgraded successfully:

```
$MOOGSOFT_HOME/bin/utils/moog_db_validator.sh
```

- Confirm that all the steps are successful. Some schema differences might be valid (e.g. `custom_info` related). If there are more substantial differences, you should investigate further to verify that all the pre-requisite upgrade scripts have been applied in the right order:

```
mysql -u root <moogdb_database_name> -e "select * from
schema_upgrades;"
mysql -u root <moog_reference_database_name> -e "select * from
schema_upgrades;"
```

Troubleshooting

If you have any issues, refer to Troubleshooting.

Upgrade - Migrate from RPM to Non-root

This topic describes the upgrade procedure from Cisco Crosswork Situation Manager v6.4.0 or v6.5.0 (RPM-based) to Cisco Crosswork Situation Manager v7.0.0 (non-root).

Fundamentally this process involves installing a fresh non-root v7.0.0 deployment of Cisco Crosswork Situation Manager onto a new server then migrating the v6.5.0 RPM- based database and configuration across.

The following steps outline this workflow:

Before You Begin

Red Hat Enterprise Linux 6/CentOS 6 is not supported for Cisco Crosswork Situation Manager v7.0.0. See Deprecation Notice: RHEL 6 . If your previous installation runs on RHEL 6, see the Upgrade - Migrate from RHEL6 to RHEL7 guide.

This guide assumes the non-root deployment is on a different host to the RPM-deployment, in order to avoid the need to completely uninstall the existing deployment before verifying the new non-root one.

Back up the Existing System

To back up the existing system:

1. Back up `$MOOGSOFT_HOME`.
2. Take a snapshot (for VMs).
3. Back up MySQL.

Stop the Existing RPM-based Services

Run the following commands to stop the existing services:

```
service apache-tomcat stop
service nginx stop
service elasticsearch stop
service moogfarmd stop
service <lam_name> stop
service rabbitmq-server stop
```

Perform a Database Dump

Run the following commands to perform a database dump:

```
mysqldump --single-transaction --set-gtid-purged=OFF --routines --triggers
moogdb > moogdb.sql
mysqldump --single-transaction --set-gtid-purged=OFF --routines --triggers
moog_reference > moog_reference.sql
mysqldump --single-transaction --set-gtid-purged=OFF --routines --triggers
historic_moogdb > historic_moogdb.sql
```

Stop the MySQL Service

Run the following command to stop the RPM-based MySQL service:

```
service mysqld stop
```

Install the Non-root Cisco Crosswork Situation Manager v7.0.0

Choose a deployment scenario for the new non-root Cisco Crosswork Situation Manager v7.0.0:

- Single Host Installation for Non-root
- Distributed Installation for Non-root

Migrate Data from the RPM Deployment into the New Non-root Deployment

Migrate the configuration files and the database.

Migrate the Configuration Files

The following configuration files/folders should be copied from their current RPM-based location and then manually merged into the new non-root version of each file on the Cisco Crosswork Situation Manager server. **The permissions MUST be set to the same as they were on the original deployment, but the ownership should be changed to the non-root user**:

- `$MOOGSOFT_HOME/config/` - You should migrate any customized files in this folder and merge them into the new ones.
- `$MOOGSOFT_HOME/bots/{lambots,moobots}` - You should migrate any customized files in these folders and merge them into the new ones.
- `$MOOGSOFT_HOME/contrib/` - You should migrate any files used by LAMs or moolets in this folder.
- `/etc/init.d/apache-tomcat` - If the `CATALINA_OPTS` string was customized in this script, you need to set the altered settings in the new `$MOOGSOFT_HOME/bin/utils/process_cntl` script on the line starting with `export CATALINA_OPTS=....`
- `/etc/init.d/{restlamd,socketlamd,traplamd,. }` - For non-root, LAMs are no longer started using service scripts. If multiple LAMs are being used (same type or different), you can start them using `process_cntl`. For this to work, you need to change the config file names. For example:

```
$MOOGSOFT_HOME/bin/utils/process_cntl socket_lam start --
service_instance=KINGSTON
```

- `/etc/nginx/conf.d/moog-ssl.conf` - If this file or any other Nginx file has been customized, you must merge it with the new one under `$MOOGSOFT_HOME/cots/nginx/config/conf.d/`. You must copy across any certificate files used in the original RPM-based deployment and you must update their paths in the new `moog-ssl.conf` file.
- `/var/lib/moogsoft/moog-data/` - You must copy across this entire directory contents to the new server into `$MOOGSOFT_HOME/moog-data/`. You must set the permissions to the same as they were on the original deployment, but you must change the ownership to the non-root user.
- `$MOOGSOFT_HOME/ui/integrations` - You must copy this entire directory contents to the new server into `$MOOGSOFT_HOME/ui/integrations`. You must set the permissions to the same as they were on the original deployment, but you must change the ownership to the non-root user. An example of a way to copy across old integrations UI files from one folder to the non-root one, which assumes the RPM-based `$MOOGSOFT_HOME/ui/integrations` folder has been copied to `/tmp/integrations/`, is shown below:

```
ROOT_INTEGRATIONS_FOLDER=/tmp/integrations;
for file in $(diff -qr $MOOGSOFT_HOME/dist/7.0.0/ui/integrations/ \
${ROOT_INTEGRATIONS_FOLDER} | egrep -vi 'only in.*7.0.0|differ' \
| awk '{print $NF}'); do cp -rp "${ROOT_INTEGRATIONS_FOLDER}/$file" \
$MOOGSOFT_HOME/dist/7.0.0/ui/integrations/; done
```

- `/etc/my.cnf` - You must make any customizations specific to the RPM-based MySQL configuration file to the new non-root MySQL configuration file: `~/.my.cnf`.

Migrate the Database

Copy the three database dump files from the original server onto the non-root server. Then import the database dumps into the new database. Replace the database names below as appropriate if they are not the default names.

```
mysql -u root moogdb < moogdb.sql
mysql -u root moog_reference < moog_reference.sql
mysql -u root historic_moogdb < historic_moogdb.sql
```

Upgrade the Cisco Crosswork Situation Manager Schema

To upgrade the Cisco Crosswork Situation Manager database, you need to provide the `moog_db_auto_upgrader` utility with the credentials of a *database* user with super privileges. For single-host installs where MySQL was installed as part of the Cisco Crosswork Situation Manager deployment, you can use the default 'root' user.

1. Run the following command after substituting the `<MySQL-SuperUsername>` argument:

```
bash $MOOGSOFT_HOME/bin/utils/moog_db_auto_upgrader -t 7.0.0 -u
<MySQL-SuperUsername>
```

2. Enter the password for that user. You can provide the password to the utility using the `-p` flag but this is not recommended in non-test deployments for security reasons.
3. Run the following commands to fix some indexing issues:

```
mysql -u root -e "use moogdb; alter table alert_filters_access drop
key \`filter_id\`"
mysql -u root -e "use moogdb; alter table situation_filters_access
drop key \`filter_id\`"
mysql -u root -e "use moogdb; alter table enrichment_static_mappings
drop key \`enrichment_static_mappings_ibfk_1\`"
mysql -u root -e "use moogdb; alter table sig_stats_cache drop key
\`sig_id\`"
```

Restart All Processes

To restart all processes:

1. Run the following commands to restart the processes:

```
$MOOGSOFT_HOME/bin/utils/process_cntl apache-tomcat restart
$MOOGSOFT_HOME/bin/utils/process_cntl moog_farmd restart
$MOOGSOFT_HOME/bin/utils/process_cntl nginx restart
$MOOGSOFT_HOME/bin/utils/process_cntl elasticsearch restart
```

2. To restart LAMs or moog_farmd instances using specific configuration files, you can use the following example commands:

```
$MOOGSOFT_HOME/bin/utils/process_cntl socket_lam start --
service_instance=KINGSTON
$MOOGSOFT_HOME/bin/utils/process_cntl moog_farmd start --
service_instance=SURBITON
```

3. Run the following command to restart any previously running UI-based Integrations:

```
bash $MOOGSOFT_HOME/bin/utils/startup_cntl;
```

Install the Integrations

Run the following command to install the UI integrations for Cisco Crosswork Situation Manager v7.0.0, if required:

```
```

```
bash $MOOGSOFT_HOME/bin/utils/integration_installer -a -l WARN;
```

Elasticsearch Re-index

Run the following command to re-index the alerts and Situations in moogfarmd. This command requires moogfarmd to be running.

```
$MOOGSOFT_HOME/bin/utils/moog_indexer --now
```

The re-index occurs in the background and may take a while to complete. This depends on the number of alerts and Situations in your system.

Migrate the Java Keystore

The upgrade includes a new version of the Java Runtime so you need to migrate any certificates stored in the old Java keystore into the new keystore, which is now the JRE under `$MOOGSOFT_HOME/cots/jre/`.

If you did not manually add any certificates, you can skip this step.

Verify the Upgrade

Run the following automatic utilities to ensure that the upgrade was successful:

1. Validate upgrade of Cisco Crosswork Situation Manager core binary files

   ```
   $MOOGSOFT_HOME/bin/utils/moog_install_validator.sh
   ```

2. Validate upgrade of apache-tomcat webapps and configuration

   ```
   $MOOGSOFT_HOME/bin/utils/tomcat_install_validator.sh
   ```

3. Validate that the database schema has been upgraded successfully

   ```
   $MOOGSOFT_HOME/bin/utils/moog_db_validator.sh
   ```

4. If there are some schema differences, they may be valid (e.g. `custom_info` related). If there are more substantial differences, you should investigate further to check if all the pre-requisite upgrade scripts have been applied in the right order:

   ```
   mysql -u root <moogdb_database_name> -e "select * from
   schema_upgrades;"
   mysql -u root <moog_reference_database_name> -e "select * from
   schema_upgrades;"
   ```

Remove the RPM-based Cisco Crosswork Situation Manager

Once the migration and upgrade have been completed successfully, you can remove the Cisco Crosswork Situation Manager packages and dependencies from the original RPM-based server, if required.

Please Please refer to the Uninstalling Cisco Crosswork Situation Manager guide for instructions on how to do this.

Troubleshooting

If you have any issues, refer to Troubleshooting.

Upgrade - Migrate from RHEL6 to RHEL7

Red Hat Enterprise Linux 6/CentOS 6 (RHEL6/CentOS6) is not supported for Cisco Crosswork Situation Manager v7.0.0. As a result, if you want to upgrade to Cisco Crosswork Situation Manager v7.0.0, you need to install the same version of Cisco Crosswork Situation Manager onto a Red Hat Enterprise Linux 7/CentOS 7 (RHEL7/CentOS7) platform and migrate your data across before upgrading to Cisco Crosswork Situation Manager v7.0.0.

This document describes the recommended process for migrating the current version of Cisco Crosswork Situation Manager from RHEL6/CentOS6 to RHEL7/CentOS7. You can then use the standard upgrade documents linked at the bottom of this page to perform the upgrade itself.

This document only covers the RPM-based Cisco Crosswork Situation Manager deployments, as the non-root version does not support RHEL6/CentOS6.

Install the Existing Version of Cisco Crosswork Situation Manager onto a New RHEL7/CentOS7 Server

- If you are using Cisco Crosswork Situation Manager v6.4.0.x, please perform a clean install of this version on the new RHEL7/CentOS7 server using the following guides:
    1. https://docs.moogsoft.com/pages/viewpage.action?pageId=18416746
    2. https://docs.moogsoft.com/display/060400/Single+Host+Installation
- If you are using Cisco Crosswork Situation Manager v6.5.0.x, please perform a clean install of this version on the new RHEL7/CentOS7 server using the following guides:
    1. https://docs.moogsoft.com/pages/viewpage.action?pageId=19247428
    2. https://docs.moogsoft.com/display/060500/Single+Host+Installation+-+RPM

Migrate the Configuration Files and Data

Migrate the Configuration Files

The following configuration files should be copied from their current location onto the same location on the new server. **The ownership and permissions MUST be set to the same as they were on the original server.**

- `$MOOGSOFT_HOME/config/` - You should migrate any customized files in this folder, as appropriate.
- `$MOOGSOFT_HOME/contrib/` - You should migrate any files used by LAMs or moolets in this folder.
- `/etc/init.d/apache-tomcat` - You should migrate this service script if it has been customized.
- `/etc/init.d/moogfarmd` - You should migrate this service script if it has been customized.
- `/etc/init.d/{restlamd,socketlamd,traplamd,...}` - If any of the default LAM service scripts have been customized or new LAM service scripts have been added to this folder, you should migrate them.
- `/etc/nginx/conf.d/moog-ssl.conf` - If this file or any other Nginx file has been customized, you should migrate it.
- `/var/lib/moogsoft/` - You should copy this entire directory across to the new server. Ensure folder permissions and ownership are the same as on the other server.
- `$MOOGSOFT_HOME/ui/integrations` - You should migrate this folder if UI-based integrations are being used.
- `/etc/my.cnf`

Migrate the Database

You only need to follow these steps if the `moogsoft-db` package is installed alongside the MySQL package on a RHEL6/CentOS6 server. To migrate the database:

1. Stop all LAMs and moogfarmd.
2. Stop Apache Tomcat.
3. Run the following commands to perform a database dump:

```
mysqldump --single-transaction --set-gtid-purged=OFF --routines --
triggers moogdb > moogdb.sql
mysqldump --single-transaction --set-gtid-purged=OFF --routines --
triggers moog_reference > moog_reference.sql
mysqldump --single-transaction --set-gtid-purged=OFF --routines --
triggers historic_moogdb > historic_moogdb.sql
```

4. Copy the three SQL dump files onto the new RHEL7/CentOS7 server.
5. Import the database dumps into the new database:

```
mysql moogdb < moogdb.sql
mysql moog_reference < moog_reference.sql
mysql historic_moogdb < historic_moogdb.sql
```

Restart MySQL

If the `my.cnf` file has been customized, restart the `mysqld` service:

```
service mysqld restart
```

Verify the Migration

To verify the migration:

1. Validate migration of Cisco Crosswork Situation Manager core binary files:

   ```
   $MOOGSOFT_HOME/bin/utils/moog_install_validator.sh
   ```

2. Validate migration of apache-tomcat webapps and configuration:

   ```
   $MOOGSOFT_HOME/bin/utils/tomcat_install_validator.sh
   ```

3. Validate that the database schema has been migrated successfully:

   ```
   $MOOGSOFT_HOME/bin/utils/moog_db_validator.sh
   ```

4. Some schema differences may be valid (e.g. `custom_info` related). If there are more substantial differences, you should investigate further.

Proceed with the Upgrade

After the migration has been completed, see Upgrade - RPM Deployments to continue the upgrade to Cisco Crosswork Situation Manager v7.0.0.

Remove Cisco Crosswork Situation Manager from the RHEL6/CentOS6 Server

Once the upgrade has completed successfully, you can remove the Cisco Crosswork Situation Manager packages and dependencies from the RHEL6/CentOS6 server, if required.

Please refer to the Uninstalling Cisco Crosswork Situation Manager guide for instructions on how to do this.

Uninstalling Cisco Crosswork Situation Manager

Introduction

This guide will provide steps should you need to uninstall Cisco Crosswork Situation Manager and its supporting packages. The aim being to revert a machine back to its pre-Cisco Crosswork Situation Manager state.

To uninstall Cisco Crosswork Situation Manager, follow the steps outlined below. Be sure to backup any files that you may need again for another installation.

Stop Core Cisco Crosswork Situation Manager and Supporting Services

1. Stop all core Cisco Crosswork Situation Manager services:

```
service moogfarmd stop
service logfilelamd stop
service restlamd stop
service socketlamd stop
service trapdlamd stop
```

2. Stop any additional moog_farmd or lam instances running as services.

```
service <service name> stop
```

3. As a precaution, forcibly kill any remaining core Cisco Crosswork Situation Manager processes:

```
kill -9 $(ps -ef|grep java|grep lam|awk '{print $2}') 2>/dev/null
kill -9 $(ps -ef|grep java|grep moog_farmd|awk '{print $2}') 2>/dev/null
```

4. Stop all supporting services:

```
service nginx stop
service elasticsearch stop
service apache-tomcat stop
service mysqld stop
service rabbitmq-server stop
```

Uninstall Core Cisco Crosswork Situation Manager Packages and Remove Directories and Users

1. Uninstall Core Cisco Crosswork Situation Manager Packages

```
yum remove $(rpm -qa|grep moogsoft)
```

2. Remove Core Cisco Crosswork Situation Manager Directories

```
rm -rf /usr/share/moogsoft
rm -rf /var/lib/moogsoft
rm -rf /var/log/moogsoft
rm -rf /var/run/moogsoft
```

3. Remove any Cisco crontab entries with this command:

```
crontab -l | egrep -v "moog|JAVA_HOME" | crontab -
```

4. Remove Cisco system users (and their home directories):

```
userdel -r moogsoft
userdel -r moogadmin
userdel -r moogtoolrunner
groupdel moogsoft
```

Uninstall Supporting Applications

Follow these steps to remove the supporting applications Apache-Tomcat, ElasticSearch, MySQL, Nginx and RabbitMQ.

Uninstalling Apache Tomcat

> **Assumption:** The Apache-Tomcat service has already been stopped as per previous instructions above

To uninstall Apache-Tomcat remove the installation directories and the service script:

> **Please note**: Apache Tomcat is not actually installed as an rpm package but is deployed as a tarball (via the moog_init_ui.sh script).

```
rm -rf /usr/share/apache-tomcat
rm -rf /var/run/apache-tomcat
rm -f /etc/init.d/apache-tomcat
```

To remove the tomcat system user and its home directory:

```
userdel -r tomcat
```

Uninstalling Elasticsearch

> **Assumption:** The Elasticsearch service has already been stopped as per previous instructions above.

To remove the Elasticsearch package:

```
yum remove elasticsearch
```

To remove related directories run the following commands:

```
rm -rf /usr/share/elasticsearch
rm -rf /var/lib/elasticsearch
```

Uninstalling MySQL

> **Assumption:** The mysqld service has already been stopped as per previous instructions above.

Remove the MySQL community packages with the following command:

```
yum remove $(rpm -qa|grep mysql)
```

To remove the related directories:

```
rm -rf /usr/share/mysql
rm -rf /var/lib/mysql
```

To remove the MySQL system user and its home directory and group:

```
userdel -r mysql
```

Uninstalling Nginx

> **Assumption:** The Nginx service has already been stopped as per previous instructions above.

Remove the nginx and supporting packages with the following command:

```
yum remove nginx
```

To remove related directories:

```
rm -rf /etc/nginx
rm -rf /usr/lib64/nginx
rm -rf /usr/share/nginx
rm -rf /var/log/nginx
rm -rf /var/lib/nginx
```

To remove the Nginx system user and its home directory and group:

```
userdel -r nginx
```

Uninstalling RabbitMQ

Assumption: RabbitMQ server service has already been stopped as per previous instructions above.

Remove the rabbitmq-server package with the following command:

```
yum remove rabbitmq-server
```

To remove related directories:

```
rm -rf /etc/rabbitmq
rm -rf /usr/lib/ocf/resource.d/rabbitmq
rm -rf /var/log/rabbitmq
rm -rf /var/lib/rabbitmq
```

To stop the erlang epmd daemon:

```
epmd -kill
```

Please note: The above command may not be necessary on EL7 installs.

To remove the RabbitMQ system user and its home directory and group:

```
userdel -r rabbitmq
```

Uninstall Remaining Packages and Remove Yum Repositories

Optionally, follow these steps to remove the remaining packages that are typically added during a Cisco Crosswork Situation Manager installation and clean up the Yum repositories:

Remove remaining packages:

Important: Note that the below list of packages is based on reverting back to a "minimal" installation of CentOS 6.9 and will vary with different versions of Linux and installation level.

Care should be taken not to remove packages that impact other important applications that may be installed on the server.

Review carefully the yum summary before proceeding with the removal - specifically any other packages listed in the "`Removing for dependencies:`" output.

In the list of removal packages below, the **libX*** and **perl*** packages may typically impact other applications.

To remove the remaining packages, run these commands:

```
yum remove GeoIP GeoIP-GeoLite-data GeoIP-GeoLite-data-extra \
apr compat-readline5 erlang fontconfig freetype gd geoipupdate jdk1.8.0
```

```
_121 \
libX11 libX11-common libXau libXpm libgfortran libjpeg-turbo libkqueue
libpng libxcb libxslt \
nginx-filesystem \
perl perl-DBI perl-Module-Pluggable perl-Pod-Escapes perl-Pod-Simple perl-
libs perl-version \
socat tomcat-native
```

Remove Yum Repositories:

Remove EPEL Yum Repository:

```
yum remove epel-release
rm -f /etc/yum.repos.d/epel*
```

Remove MySQL Community Yum Repository:

```
yum remove mysql-community-release
rm -f /etc/yum.repos.d/mysql*
```

Remove remaining Yum Repositories:

```
rm -f /etc/yum.repos.d/elasticsearch.repo
rm -f /etc/yum.repos.d/moog.repo
rm -f /etc/yum.repos.d/rabbitmq_rabbitmq-server.repo
```

yum remove $(rpm -qa|grep mysql)

Change Passwords for Default Users

Cisco Crosswork Situation Manager creates systems users for Linux, MySQL, and RabbitMQ during the installation process. As a security measure, you can change the password for these users. After you change the passwords for certain users, you must update Cisco Crosswork Situation Manager configuration to use the new password.

If you run in a distributed environment, you can set unique passwords for all components on each host. Update the configuration files for a host to contain the password for user for the host.

Cisco recommends you encrypt passwords for use in Cisco Crosswork Situation Manager configuration files. See Moog Encryptor. In distributed or high availability environments, encrypt the password using the Moog Encryptor on each machine.

Linux Users

The Cisco Crosswork Situation Manager installation package creates the following

- Linux users with login privileges: moogsoft
- moogadmin
- rabbitmq
- tomcat
- moogtoolrunner

Execute the `passwd`command to change the password Linux users. For example, to change the password for `moogtoolrunner`:

```
```

```
passwd moogtoolrunner
```

Update Configuration

After you change the password for moogtoolrunner, update its password in `$MOOGSOFT_HOME/config/servlets.conf`. For example:

```
# The toolrunner user password.
        # Use either toolrunnerpassword or toolrunnerpassword.
        toolrunnerpassword: "MyNewPassword"
        # encrypted_toolrunnerpassword: "rmW2daCwMyI8JGZygfEJj0MZdbIkUqX3tT
/OIVfMGyI=",
```

Restart Apache Tomcat to apply the configuration change.

```
service apache-tomcat restart
```

> You do not need to update the Cisco Crosswork Situation Manager configuration after you change the password for other Linux users with login privileges.

Other Linux Users

The Cisco Crosswork Situation Manager installation package creates the following

- users without login privileges: elasticsearch
- mysql
- nginx

MySQL User

The Cisco Crosswork Situation Manager installation process creates a user to log in to MySQL: ermintrude. Execute the MySQL `SET PASSWORD` statement for the user ermintrude to change its password:

```
SET password FOR 'ermintrude'@'localhost'= PASSWORD('<new-password>');
```

If you are running in a distributed environment, update the password for the ermintrude user on all other nodes:

```
SET password FOR 'ermintrude'@'<host>'= PASSWORD('<new-password>');
```

Where `<host>` is the name of the remote server connecting to the database. For example, if you have the Cisco Crosswork Situation Manager UI web components running on a separate host named "mywebserver":

```
SET password FOR 'ermintrude'@'mywebserver'= PASSWORD('MyNewPassword');
```

After you change the password for ermintrude, grant it priviliges on all the objects in the `moogdb` and `moog_reference` databases. For example:

```
GRANT ALL ON moogdb.* TO ermintrude@'localhost' IDENTIFIED BY '<new-
password>';
GRANT ALL ON moog_reference.* TO ermintrude@'localhost' IDENTIFIED BY
'<new-password>';
```

If you are running in a distributed environment, you must grant permissions for the ermintrude user on all nodes. For example:

```
GRANT ALL ON moogdb.* TO ermintrude@'<host>' IDENTIFIED BY '<new-
password>';
GRANT ALL ON moog_reference.* TO ermintrude@'<host>' IDENTIFIED BY '<new-
password>';
```

Where `<host>` is the name of the remote server connecting to the database. For example, if you have the Cisco Crosswork Situation Manager UI web components runing on a separate host named "mywebserver":

```
GRANT ALL ON moogdb.* TO ermintrude@'my' IDENTIFIED BY 'MyNewPassword';

GRANT ALL ON moog_reference.* TO ermintrude@'<host>' IDENTIFIED BY 'MyNewPassword';
```

Update Configuration

After you change the password for ermintrude, update its password in `$MOOGSOFT_HOME/config/system.conf`. For example:

```
    "mysql" :
        {
            "host"            : "localhost",
            # The name of the moogdb database
            "moogdb_database_name"          : "moogdb",
            # The name of the moog_reference database
            "referencedb_database_name"     : "moog_reference",
            "username"        : "ermintrude",
            # "encrypted_password":"vQj7/yom7e5ensSEb10v2Rb/pgkaPK
/4OcUlEjYNtQU=",
            "password"        : "MyNewPassword"
```

If you are running in a distributed environment, update the password configuration on every host.

RabbitMQ User

The Cisco Crosswork Situation Manager installation process creates a RabbitMQ user called moogsoft. Execute the `rabbitmqctl change_password` command to change the password for moogsoft. For example:

```
rabbitmqctl change_password moogsoft <new-password>
```

Update Configuration

After you change the password for moogsoft, update its password in `$MOOGSOFT_HOME/config/system.conf`. For example:

```
      # By default the moogsoft username and password are used.
        # This needs to match the MooMS broker configuration. If
        # commented out a default "guest" user will be used.
        #
        "username"        : "moogsoft",
```

```
        "password"        : "MyNewPassword",
        # "encrypted_password" : "e5uO0LY3HQJZCltG
/caUnVbxVN4hImm4gIOpb4rwpF4=",
```

If you are running in a distributed environment, update the password configuration on every host.

High Availability

High Availability (HA) deployments of Cisco Crosswork Situation Manager comprise multiple instances of Cisco Crosswork Situation Manager, moogfarmd, and the associated LAMs to minimize downtime and data loss. Component redundancy protects against single points of failure. It also provides reliable mechanisms to enable failover from one component to another to avoid performance degradation and data loss.

See HA - Deployment Scenarios for deployment examples.

HA Components

Cisco Crosswork Situation Manager is made up of the following set of processes and services components, which can be implemented in a distributed environment:H A in Cisco Crosswork Situation Manager

- Event ingestion (LAMs)
- Event processing (moog_farmd)
- User interface (Nginx, servlets running in Tomcat and Elasticsearch)
- RabbitMQ broker (MooMS messaging system)
- Database (MySQL 5.6)

See HA - Setup for Dependencies for more information on how to set these up for distributed installations.

Introducing component redundancy (for example two identically configured event processing (moog_farmd) components) makes HA system architecture possible. Implementing HA system architecture enables failover of Cisco Crosswork Situation Manager components without loss of data or performance.

Failover of Cisco Crosswork Situation Manager components is manually triggered using the ha_cntl command line utility which also allows the status of all components in the HA installation to be viewed.

Cisco Crosswork Situation Manager HA Features at a Glance

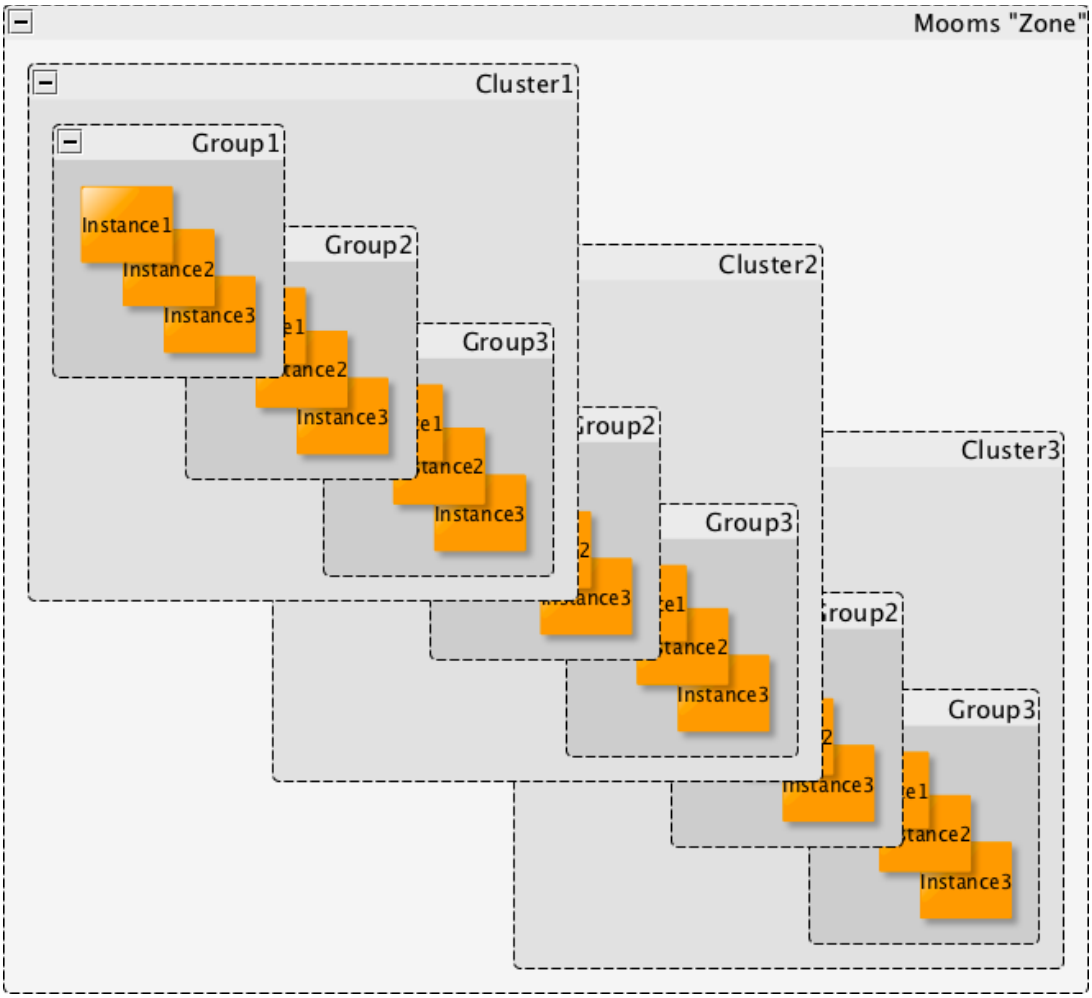
- LAMs, moog_farmd and Tomcat servlets can run in 'Active' mode (normal operation) or 'Passive' mode (not processing messages)
- Instance/Process Group/Cluster naming convention for LAMs, moog_farmd and Tomcat servlets to build logical groupings for failover scenarios
- ha_cntl utility to show running HA components and to trigger manual failovers (by setting Instances/Process Groups /Clusters to Active or Passive)
- 'Leader' capability to allow only defined Instances to become Active when their parent Cluster/Process Group becomes Active
- Moolet state sharing ability (persistence) for moog_farmd to facilitate data integrity during failover of moog_farmd
- MySQL 'failover' connection definition to allow Cisco Crosswork Situation Manager components to failover to backup MySQL servers if the primary connection goes down
- Handling for the UI to continue normal operation in the event of a UI failover
- Self Monitoring pages in UI and moog_monitor command line utility show HA information
- Product installation using split RPMs (by functional component) for easier distributed deployment

Cisco Crosswork Situation Manager HA architecture key concepts

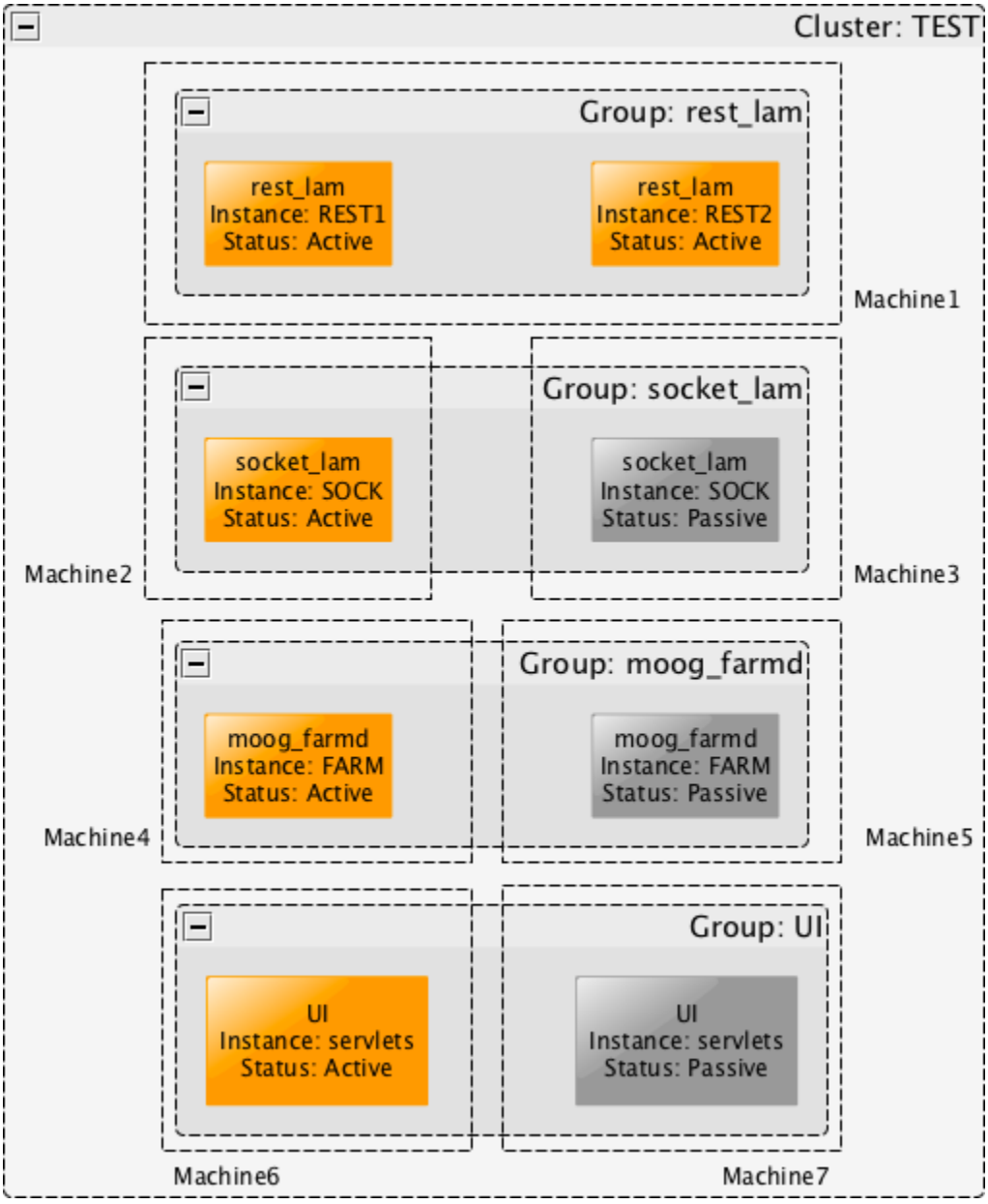
Concept	Description
Component	An instance of a Cisco Crosswork Situation Manager LAM, servlet or moog_farmd. HA introduces redundancy on a component, Process Group or Cluster level.
Instance	A name for each Cisco Crosswork Situation Manager component.
Process Group	A group of one or more of the same type of Cisco Crosswork Situation Manager components (such as a group of load sharing socket LAMs). All moog_farmd components in a Process Group must have identical configuration.
Cluster	One or more Process Groups. A Cluster must contain at least one

	Process Group.
Zone	Any number of Clusters, Process Groups and Instances can be defined within a single MooMS 'Zone' (RabbitMQ broker vhost). Failover actions for those Clusters/Process Groups/Instances are limited to be within that Zone.

Cisco Crosswork Situation Manager Architecture



Instances are individual components that run on a single machine. Process Groups and Clusters, however, can span multiple machines. Their configuration allows the flexibility to define architectural groupings for failover actions as long as they are within the same MooMS Zone:



Component	Description
Active / Passive mode	Instances are configured to operate in Active or Passive mode. Instances that are operational are set to Active mode. Instances that are backup (redundant components) operate in Passive mode
Leader	Defines which Instance in a Process Group becomes Active when the whole Process Group switches from Passive to Active state. Normally, only one Instance per Process Group should be defined as a Leader. Leader definition availability is as follows:

Component	Leader definition availability
moog_farmd	Mandatory for a Process Group with more than one moog_farmd
Socket LAM, Logfile LAM,TrapdLAM	Optional
REST LAM, UI	Not applicable

servlets	

Leadership status is a property of Process Groups. There are two states of group leadership status (as seen in the output of `ha_cntl --view` command. See below) as follows:

State	Description
"only leader should be active"	This is the default setting for components where Leader definition is supported (in moog_farmd, Socket LAM, Logfile LAM, Trapd LAM)
"no leader - all can be active"	This is the default setting for components where Leader definition is not supported (REST LAM and the UI servlets), OR if there is one or more Instances in the Process Group configured with `"only_leader_active = false,"`. The behavior is dynamic, i.e. terminating such an Instance will change back the Process Group's status to `"only leader should be active"`

All of the above are defined when starting each component. If any of the values are not explicitly defined as parameters at startup, values are taken from the component configuration file (or if not defined there, values in `system.conf` are used).

Example component definition

```
$MOOGSOFT_HOME/bin/moog_farmd --cluster Surbiton --instance MASTER --
leader yes --mode active
```

The above creates an Instance of `moog_farmd` and defines it as a member of the `Surbiton` Cluster, with the Instance name `MASTER`. It also defines it as the Leader Instance in its Process Group and configures it to operate in Active mode. No Process Group (`--group`) is defined, so the default name (from the component configuration file) `moog_farmd` is used.

Cisco Crosswork Situation Manager HA Configuration

The information in the table describes how to configure Cisco Crosswork Situation Manager components for a HA architecture.

<table>
<tr><th>Component</th><th>File</th><th>Section</th><th>Example</th><th>Description</th></tr>
<tr><td>Default Cluster</td><td>$MOOGSOFT_HOME/config/system.conf</td><td><code>ha section, cluster property</code></td><td><pre>"ha":
{
 "cluster": "NY"
}</pre></td><td>The name of the Cluster. This supersedes anything set in <code>system.conf</code> (can also be overwritten by the command line)</td></tr>
<tr><td>LAMs</td><td>LAMs configuration file</td><td><code>ha</code> section</td><td><pre>ha:
{
  cluster: "NY",
  group: "socket_lam",
  default_leader: true,
  only_leader_active: true,

accept_conn_when_passive: true
}</pre></td><td><table><tr><th>Property</th><th>Description</th></tr><tr><td>cluster</td><td>The name of the Cluster. This supersedes anything set in <code>system.conf</code> (can also be overwritten by the command line)</td></tr></table></td></tr>
</table>

				group	The name of the Process Group. This defaults to the LAM process name if no value is specified (for example `socket_lam`)
				default_leader	A Boolean, indicating if the LAM is the Leader within its Process Group (see above). The default value is `true` if not specified
				only_leader_active	A Boolean that changes the type of Process Group from a Leader Only group to a Process Group where more than one process can be Active. The default is `true`, except for the REST LAM where it is not supported and it is always treated as `false`
				accept_conn_when_passive	A Boolean instructing the LAM what to do in Passive mode. If `true` (or not set), the LAM accepts incoming connections but discards any events received. If `false`, the LAM does not accept incoming connections, and closes the socket from socket/Trapd LAMs. This is to prevent a load balancer from detecting them as unavailable and routing traffic elsewhere
moog_farmd	`moog_farmd.conf`	`ha` section			

			`ha:`
			`{`
			`  cluster: "NY",`
			`  group: "moog_farmd",`
			`  default_leader: true,`
			`}`
				Property / **Description**
				cluster: The name of the Cluster. This supersedes anything set in `system.conf` (can also be overwritten by the command line)
				group: The name of the Process Group. This defaults to `moog_farmd`
				default_leader: A Boolean, indicating if this moog_farmd is the Leader within its Process Group (see above). Defaults to `true` if no value is specified

Command line overwrites	**Component** / **Description** / **Command line** Cluster: `cluster SF` to the command line for starting the component — `cluster SF` Process Group: `group cool_group` to the command line for starting the component — `group cool_group` Instance: `instance instance_3` (for example) to the command line for starting this Instance of the component — instance instance_3 Passive Mode — `mode passive` to the command line for starting this Instance of the component Instance not Process Group leader (where `only_leader_active` is set) — `leader no` to the command line for starting this Instance of the component. This will overwrite the `default_leader` in the configuration files			

	Example:
	`$MOOGSOFT_HOME/bin/moog_farmd --instance TEST_INSTANCE --group TEST_GROUP --cluster TEST_CLUSTER --mode passive`
	`$MOOGSOFT_HOME/bin/socket_lam --instance SOCK1 --group SOCKGROUP --cluster CLUSTER1 --leader no --mode passive`			

Servlets	$MOOGSOFT_HOME/config/servlets.conf	`ha` section	`ha :`
			`{`
			`  instance: "servlets",`
			`  group: "UI",`
			`  start_as_passive: false`
			`}`
				• Note that all servlets defined in this file act as one HA "instance" - hence will all failover together
				• If cluster is not specified, the name of the Cluster is taken from the system.conf file
				• If group is not specified, the name defaults to "servlets"
				• If start_as_passive is not specified, then the servlet

				defaults to a setting of false for this property; hence, it is Active on startup

- Servlets do not support any 'leader' settings
- The apache-tomcat service must be restarted to apply configuration changes made to the servlets

Active and Passive mode behavior

In a High Availability deployment, Cisco Crosswork Situation Manager components may operate in either Active or Passive mode. In Active mode, their behavior is unchanged from non-HA Cisco Crosswork Situation Manager installations, carrying out data ingestion, processing, presentation, etc. In Passive mode, these activities do not occur - the component is effectively on standby, waiting for an instruction to start the processing activities defined by its component type and configuration setup. Failover is the process of converting one or more processes from Active to Passive mode while converting other processes from Passive to Active mode.

The Active/Passive state of the HA components in a Cluster can be viewed in the Cisco Crosswork Situation Manager UI using Self Monitoring or via the ha_cntl utility (see below). In the UI, Passive processes are indicated by the icon.
Further details of how components behave in Passive mode and how, where relevant, the Passive mode may be identified from the command line are given below.

Servlets

When the UI is in Passive mode, the moogsvr servlet will reject all requests with an HTTP status of `503` (server unavailable) and the moogpoller servlet will not accept incoming websocket upgrade requests. When switching from active to passive the moogsvr servlet will start rejecting requests and the moogpoller servlet will disconnect any existing websocket sessions.

A Load balancer can therefore determine whether a UI is running in Active or Passive mode by sending a GET request to `https://<server>:<port>/moogsvr/hastatus`. A `204` response indicates that the UI is Active, a `503` response indicates Passive mode.

The following example curl command can be sent from the command line to check servlet status:

```
curl -k https://moogbox2/moogsvr/hastatus -v
```

The output is `< HTTP/1.1 204 No Content` if the servlet is in Active mode, or `< HTTP/1.1 503 Service Unavailable` if the servlet is in Passive mode.

moog_farmd

A moog_farmd process running in Passive mode will not process events or detect Situations. When it fails over to Active mode, it will be able to carry on using the state from the previously Active Instance if this has been persisted (see below).

When the moog_farmd state is being persisted, only one moog_farmd process is allowed to run in Active mode at any given time within a single moog_farmd Process Group. If more than one moog_farmd process is started in Active mode, all but the first to become Active will be automatically converted to run in Passive mode within a few seconds. The same applies to new moog_farmd processes started in Active mode when an Active moog_farmd is already running. This prevents a condition known as 'split brain'; where two Active processes both believe that they are responsible for executing functionality.

All Instances of moog_farmd within the same Process Group must have identical configuration

LAMs

LAMs operating in Passive mode do not send Events to the MooMS bus. The REST LAM in Passive mode will reject POST requests with an HTTP status of `503` (server unavailable).

Example curl command to check rest_lam status:

```
curl -x POST http://moogbox2:9876 -v
```

The output is `< HTTP/1.1 503 Service Unavailable` if the rest_lam is in Passive mode. If the rest_lam is in Active mode, then the response code is dependent on the format of data sent to it as per normal rest_lam behavior.

Configuring persistence of state in moog_farmd

The state of moog_farmd can be persisted to ensure that context is not lost when failover occurs from one Instance of moog_farmd to another. This means that information held in memory about the Situations created by the Sigalisers and the current state of the Sigalisers themselves will not be lost. The new Instance of moog_farmd will continue to process events and detect the same Situations as would have been detected if there had been no failover.

The state of the in-memory database (and the Constants module) will always be persisted if persistence is turned on. For each of the following Sigalisers:

- Classic Sigaliser
- Speedbird
- Nexus
- Cookbook
- Template Matcher

The `persist_state` configuration parameter in `moog_farmd.conf` must be set to `true` to ensure that the state for each Sigaliser is persisted.

The state of the Alert Rules Engine Moolet can also be persisted using the `persist_state` configuration parameter. Similarly, setting "persist_state" for the AlertBuilder (or any other moolet) ensures that any tasks queued for that moolet - in this case Events that have not yet been processed - are persisted to Hazelcast while queueing and will be processed by another instance of farmd after failover.

When failover occurs, events and other pieces of information may be queued in Moolets, waiting to be processed. To ensure that these tasks are processed in the newly Active Instance of moog_farmd after failover, the `persist_state` flag is again used. This flag may be used for any Moolet that has a queue of tasks awaiting processing which, for all practical intents and purposes, is every Moolet other than the Scheduler.

> To take advantage of this feature and to ensure that the newly Active moog_farmd Instance takes over from where the previous one left off, the `message_persistence` property in the MooMS section of the `system.conf` file must be set to `true`

Choice of persistence mechanism and configuration

Persistence may be carried out using a Hazelcast in-memory Cluster. The persistence mechanism is configured in `system.conf` in the `persistence` section, for example:

```
  # Persistence configuration parameters.
    "persistence" :
        {
            # Set persist_state to true to turn persistence on. If set,
state
            # will be persisted in a Hazelcast cluster.
            "persist_state" : true,

            # Configuration for the Hazelcast cluster.
            "hazelcast" :
                {
                    # The port to connect to on each specified host.
                    "network port"      : 5701,

                    # If set to true Hazelcast will increment the port
number to
                    # an available one if the configured port is
unavailable.
                    "auto increment"    : true,
```

```
                    # A list of hosts to allow to participate in the
cluster.
                    "hosts" : ["localhost"],

                    # Additional config to allow cluster info to be viewed
via
                    # Hazelcast's Management Center UI, if running.
                    "man_center"    :
                        {
                            "enabled"   : false,
                            "host"      : "localhost",
                            "port"      : 8091
                        }
                },
...
                }
```

and as previously mentioned, ensure that the `message_persistence` property in the MooMS section of the `system.conf` file is set to `true`:

```
"mooms":
{
        "zone": "MOOG",
        "brokers": [
      {
                "host": "localhost",
                "port": 5672
          }
    ],
        "username": "moogsoft",
        "password": "m00gs0ft",
        "message_persistence": true,
        "max_retries": 100,
        "retry_interval": 200,
        "cache_on_failure": false,
        "cache_ttl": 900
}
```

Clearing Persistence Data on Start-up

If persistence is configured, once all moog_farmd Instances have been stopped, the in-memory persistence data is lost.

moog_farmd also has a command line option `--clear_state` which, when specified at start-up, clears any current persistence data for the Process Group that the moog_farmd Instance is a member of. This ensures a clean start for that particular Instance (i.e it would have no memory of previously created Situations) but also impacts any other running moog_farmd Instances in that Process Group.

> This option does not remove moog_farmd persistence data from other Process Groups

```
[root@moogbox2 regression-tests]# moog_farmd --help
\n-------- Copyright MoogSoft 2012-2015 --------\n\n  Executing:
```

```
moog_farmd\n\n------------ All Rights Reserved ------------\n
usage: moog_farmd [ --config=<path to config file> ] [ --loglevel
                  (INFO|WARN|ALL) ] [--clear_state] [ --instance <name> [
                  --cluster <name> --group <name> [ --mode
                  <passive|active> ] [ --leader <yes|no> ] ] ] [ --version
                  ]

MoogSoft moog_farmd: Container for our herd of moolets
    --clear_state      Clears any persisted state information associated
                       with this process group on startup.
    --cluster <arg>    Name of HA cluster (to overwrite the config file)
    --config           Specify a full path to the configuration file of
                       this farmd
    --group <arg>      Name of HA group (to overwrite the config file)
    --instance <arg>   Give this farmd herd a name for use with farmd
                       control
    --leader <arg>     Is this instance an HA leader within its group
                       (yes, no)
    --loglevel <arg>   Specify (INFO|WARN|ALL) to choose the amount of
                       debug output - warning ALL is very verbose!
    --mode <arg>       Start the process in passive or active mode
                       (default will be active)
    --version          Return current version of the Moog software
```

Configuring Automatic Failover for moog_farmd

When configured in an active/passive HA configuration **moog_farmd**, has the capability for **automatic failover**. This allows a passive moog_farmd to automatically take over processing from another (active) moog_farmd **in the same HA process group** if the passive moog_farmd detects that the active moog_farmd has become inactive and is failing to report its status.

This feature is controlled by three configuration properties:

- **automatic_failover**
- **keepalive_interval**
- **margin**

These properties are in the "**failover**" block in $MOOGSOFT_HOME/config/system.conf:

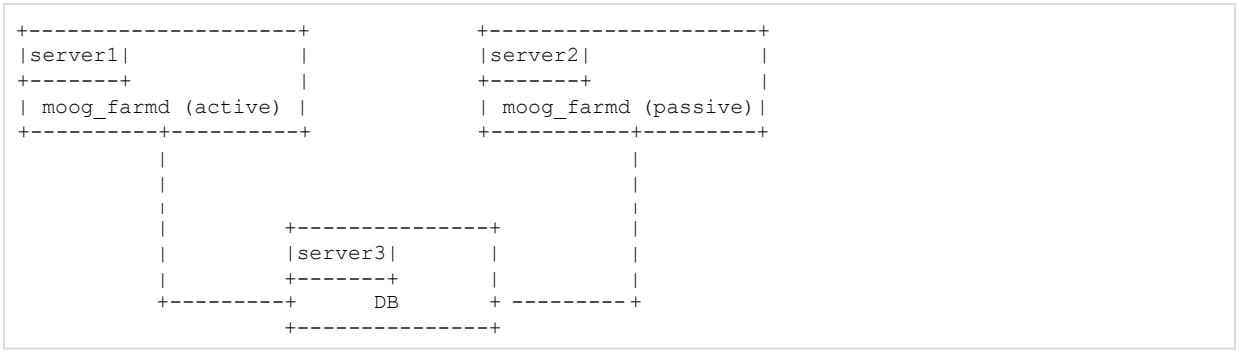
```
...
,
"failover" :
{
    "persist_state" : false,
    # Configuration for the Hazelcast cluster.
    "hazelcast" :
    {
            ...
    },
    # Failover configuration below currently applies only to moog_farmd

    # Interval (in seconds) at which processes report their
    # active/passive status and check statuses of other processes.
    "keepalive interval" : 5,
```

```
    # At next keepalive_interval, processes will allow <margin> seconds
    # before treating active processes who have not reported their
    # status as being dead.
    "margin" : 3,

    # Number of seconds to wait for previously active process to
    # become passive during manual failover. After this time has
    # expired the new instance will become active and force the
    # process to become passive.
    "failover_timeout" : 10,

    # Allow a passive process to automatically become active if
    # no other active processes are detected in the same process group
    "automatic failover" : false,

    # Process will stop indicating that it is active if it fails
    # to send <value> consecutive heartbeats.
    "heartbeat failover after": 2
},
...
```

Property	Description	Other
automatic_failover	property enables or disables the feature	true|false
keepalive_interval	(seconds and defaults to 5) defines how often a moog_farmd process reports its active/passive status to the database and checks the status of other reporting moog_farmd processes	inactive
margin	(seconds and defaults to 3) defines how long after a passive moog_farmd has detected that a formerly active moog_farmd (in its same process group) is no longer reporting status and should therefore become active and takeover processing	inactive
failover_timeout	(seconds and defaults to 10)	active
heartbeat_failover_after	(number and defaults to 2) defines that the p rocess will stop indicating that it is active if it fails to send <value> consecutive heartbeats	active

Example Automatic Failover Tuning

Assuming a highly simplified multi-host HA setup such as:



The moog_farmds on server1 and server2 are in the same process group but in different clusters. All other config is identical.

- With **automatic_failover**: false, set in system.conf, on both server1 and server2, then if the active moog_farmd process on server1 is killed, becomes unresponsive, loses contact with the DB or drops off the network, then moog_farmd on server2 will remain passive and not take over processing unless a manual failover is triggered using ha_cntl
- With **automatic_failover**: true, set in system.conf, on both server1 and server2 and with default **keepalive_interval** and **margin** settings, then if the active moog_farmd process on server1 is killed*, becomes unresponsive, loses contact with the DB or drops off the network, then moog_farmd on server2 will automatically become active and take over processing between 3-8 seconds later (depending on when next **keepalive_interval** occurs). If the moog_farmd on server1 is then restarted, resumes processing or rejoins the network, it will establish that there is already another active moog_farmd running in its process group (i.e. the instance now active on server2) and it will become passive to prevent split-brain processing occurring

Thus, the **keepalive_interval** and **margin** properties can be used to tune the sensitivity of automatic failover. In the above example (and with default settings) automatic failover happens promptly. Users may wish to increase or decrease the interval at which moog_farmd reports its status and also allow more time before a passive moog_farmd tries to take over processing (possibly useful if the active moog_farmd suffered a short interruption but has quickly resumed). Setting (for example) automatic_failover : true, keepalive_interval : 3 and margin : 10 would mean for the above system:

- the active moog_farmd process on server1 is killed*, becomes unresponsive, loses contact with the DB or drops off the network, then moog_farmd on server2 will automatically become active and take over processing between 10-13 seconds later (depending on when next keepalive_interval occurs). If the moog_farmd on server1 resumes processing or rejoins the network within 10secs of the passive moog_farmd on server2 detecting it as down, then it will continue as the active instance and the moog_farmd on server2 will remain as passive and not take over. Conversely if the moog_farmd on server1 had been restarted instead then it would not continue as the active process and the passive moog_farmd on server2 would become active and take over

* see note below on failover behaviour when process is killed or shutdown "cleanly".

Important Notes and Limitations:

- For **moog_farmd** only, automatic failover of LAMs or UI Servlets is not part of this implementation
- Identical configuration is needed on all servers running as part of the HA setup (as per other HA configuration)
- **time sensitive:** requires all servers to be time synchronised. A change to the system time on the DB server in a running HA setup could trigger automatic failover between moog_farmd instances
- Requires communication with the DB. If the DB or DB server becomes unresponsive to all moog_farmd instances then the feature will not work as expected
- If, in an automatic failover setup, an active moog_farmd instance is shutdown cleanly (i.e. using normal kill, service stop or ctrl-c) then a passive moog_farmd will take over processing at its next **keepalive_interval** and **will not wait the additional <margin> seconds**

A Note on Process Startup

If **automatic_failover** is **enabled** and a moog_farmd instance is started in passive mode and no other active moog_farmd is running in its process group, it will switch to active. Users may wish to factor this in when starting up a system i.e. it is easiest to startup active moog_farmd instances first.

A Note on Split-Brain Handling

HA implementation has built in handling to prevent split-brain processing occurring i.e. two active moog_farmds (in the same process group) running at the same time and potentially leading to duplicate processing. At its simplest it prevents a second moog_farmd being started in **active** mode if there is another **active** instance already running in the same process group (regardless of cluster or instance name). The second moog_farmd will startup but will immediately switch to **passive** mode.

Controlling Cisco Crosswork Situation Manager HA (ha_cntl)

Cisco Crosswork Situation Manager includes a High Availability Control utility to control the HA architecture. Use the ha_cntl utility to:

- failover (change status of) Instances, Process Groups or Clusters
- view the current status of all Instances, Process Groups and Clusters

There is also help available for the ha_cntl utility.

> The UI will not continue to function correctly after a failover if only one of the Tomcat servlets is failed over using activate or deactivate commands at a servlet Process Group level. Currently, the UI must be failed over at a Cluster level to ensure continued smooth operation

ha_cntl utility commands are as follows:

Command	Description
`-a,--activate <arg>`	Specify `cluster[.group[.instance_name]]` to activate Process Groups within a Cluster, a specific Process Group w Cluster or a single Instance
`-d,--deactivate <arg>`	Specify `cluster[.group[.instance_name]]` to deactiv Process Groups within a Cluster, a specific Process Group w Cluster or a single Instance
`-h,--help`	Print help text, that describes ha_cntl commands
`-l,--loglevel <arg>`	Specify `(INFO|WARN|ALL)` to choose the amount of debug
`-t,--time_out <arg>`	Specify an amount of time (in seconds) to wait for the last an not set, the default is 2 seconds
`-v,--view`	View the current status of all Instances, Process Groups and Clusters
`-y,--assumeyes`	Answer yes for all prompts. Useful for automation

Examples

Command line	Description
`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON.socket_lam.SOCK1`	This activates the socket_lam Instance `SOCK1` within the Pr Group `socket_lam` within the Cluster `SURBITON`. If the `socket_lam` Process Group is configured to be lead (see above) all other socket LAMs in the Process Group are deactivated
`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON`	This activates all Process Groups in the `KINGSTON` Cluster deactivates all other Clusters
`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.rest_lam -y`	This activates the `rest_lam` Process Group in the `KINGST` ter (and the `-y` means there is no 'are you sure?' prompt) an deactivates all other `rest_lam` Process Groups in all other
`$MOOGSOFT_HOME/bin/ha_cntl -d RICHMOND`	This deactivates all Process Groups in the `RICHMOND` Clust
`$MOOGSOFT_HOME/bin/ha_cntl -d RICHMOND.trapd_lam`	This deactivates the `trapd_lam` Process Group in the `RIC` Cluster
`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.UI -y`	This activates the UI group in the KINGSTON Cluster and wi deactivate the UI group in all other Clusters - triggering a UI (of all servlets) to the KINGSTON Cluster.

`$MOOGSOFT_HOME/bin/ha_cntl -v`	The `-v` option prints detailed status about all C Groups and Instances that it can discover:
	`[root@moogbox2 ~]# ha_cntl -v`
	`Getting system status`
	`Cluster: [KINGSTON] passive`
	`Process Group: [UI] Passive (no leader - all can be active)`
	`Instance: [servlets] Passive`
	
	`Component: moogpoller - not running`

```
Component: moogsvr - not
running

Component: toolrunner -
not running
        Process Group:
[moog_farmd] Passive
(only leader should be
active)
            Instance:
FARM Passive Leader
                Moolet:
AlertBuilder - not
running (will run on
activation)
                Moolet:
AlertRulesEngine - not
running (will run on
activation)
                Moolet:
Cookbook - not running
(will run on activation)
                Moolet:
Nexus - not running
                Moolet:
Sigaliser - not running
                Moolet:
Speedbird - not running
(will run on activation)
                Moolet:
TemplateMatcher - not
running
        Process Group:
[rest_lam] Passive (no
leader - all can be
active)
            Instance:
REST2 Passive
        Process Group:
[socket_lam] Passive
(only leader should be
active)
            Instance:
SOCK2 Passive Leader
Cluster: [SURBITON] active
        Process Group:
[UI] Active (no leader -
all can be active)
            Instance:
[servlets] Active

Component: moogpoller -
running
```

```
Component: moogsvr -
running

Component: toolrunner -
running
        Process Group:
[moog_farmd] Active (only
leader should be active)
            Instance:
FARM Active Leader
                Moolet:
AlertBuilder - running
                Moolet:
AlertRulesEngine - running
                Moolet:
Cookbook - running
                Moolet:
Default Cookbook - running
                Moolet:
Nexus - not running
                Moolet:
Sigaliser - not running
                Moolet:
Speedbird - running
                Moolet:
TemplateMatcher - not
running
        Process Group:
[rest_lam] Active (no
leader - all can be
active)
            Instance:
REST1 Active
        Process Group:
[socket_lam] Active (only
leader should be active)
            Instance:
SOCK1 Active Leader
```

farmd_cntl changes for HA

The farmd_cntl utility has 2 changes for HA:

To send farmd_cntl commands to a specific moog_farmd Instance within an HA environment, the `<Cluster>.<Process Group>.<Instance>` notation should be used for the `--instance` option. For example:

```
farmd_cntl --instance SURBITON.moog_farmd.FARM --moolet AlertBuilder --
start
```

farmd_cntl now also gives more feedback on the results of the operation(s) requested:

```
[root@moogbox2 ~]# farmd_cntl --instance CLUSTER1.GROUP1.FARM1 --all-
moolets --stop
Response from: CLUSTER1.GROUP1.FARM1
Status: Action(s) completed successfully.
Report:
    Moolet AlertBuilder Stopped.
    Moolet Default Cookbook Stopped.
    Moolet Sigaliser Stopped.
    Moolet SituationMgr Stopped.
    Moolet Cookbook Stopped.

[root@moogbox2 ~]# farmd_cntl --instance CLUSTER1.GROUP1.FARM1 --all-
moolets --stop
Response from: CLUSTER1.GROUP1.FARM1
Status: Action(s) completed with failures.
Report:
No Moolets to stop...

[root@moogbox2 ~]# farmd_cntl --instance CLUSTER1.GROUP1.FARM1 --all-
moolets --start
Response from: CLUSTER1.GROUP1.FARM1
Status: Action(s) completed with failures.
Report:
    Moolet AlertBuilder Started.
    Moolet Speedbird Started.
    Moolet Default Cookbook Started.
    Moolet TokenCounter could NOT be started.
    Moolet Sigaliser Started.
    Moolet Nexus Started.
    Moolet TemplateMatcher Started.
    Moolet AlertRulesEngine Started.
    Moolet SituationMgr Started.
    Moolet Cookbook Started.
    Moolet Notifier Started.

[root@moogbox2 ~]# farmd_cntl --instance CLUSTER1.GROUP1.FARM1 --moolet
Sigaliser --restart
Response from: CLUSTER1.GROUP1.FARM1
Status: Action(s) completed successfully.
Report:
    Moolet Sigaliser Stopped.
    Moolet Sigaliser Started.

[root@moogbox2 ~]# farmd_cntl --instance CLUSTER1.GROUP1.FARM1 --moolet
Sigaliser --moolet Speedbird --restart --reconfig
Response from: CLUSTER1.GROUP1.FARM1
Status: Action(s) completed successfully.
Report:
    Moolet Sigaliser Stopped.
```

```
Moolet Sigaliser Configuration Reloaded.
Moolet Sigaliser Started.
Moolet Speedbird Stopped.
Moolet Speedbird Configuration Reloaded.
Moolet Speedbird Started.
```

MySQL failover for Cisco Crosswork Situation Manager components

Cisco Crosswork Situation Manager allows the definition of a list of MySQL servers that, in the event that the primary connection (as defined in `mysql.host`) goes down, Cisco Crosswork Situation Manager components that have a MySQL connection (moog_farmd, tomcat, rest_lam) will automatically connect to the next available MySQL server in the `failover_connections` list.

This is defined in the `failover_connections` section of the `$MOOGSOFT_HOME/config/system.conf` file, as follows:

```
"mysql" :
        {
            "host"            : "localhost",
            "database"        : "moogdb",
            "username"        : "ermintrude",
            "password"        : "m00",
            "port"            : 3306
            #
            # New deadlock retry configuration - default values are as
below if
            # the config remains commented out.
            #
            # "maxRetries"      : 5,
            # "retryWait"       : 10
            #
            # To use Multi-Host Connections for failover support use:
            #
            #  "failover_connections" :
            #    [
            #      {
            #          "host"  : "193.221.20.24",
            #          "port"  : 3306
            #      },
            #      {
            #          "host"  : "143.47.254.88",
            #          "port"  : 3306
            #      },
            #      {
            #          "host"  : "234.118.117.132",
            #          "port"  : 3306
            #      }
            #    ]
            #
        },
```

This is useful when the system is used with a replicated/clustered MySQL environment.

Example

For the following `mysql` section in `system.conf`:

```
"mysql" :
        {
            "host"            : "moogbox1",
            "database"        : "moogdb",
            "username"        : "ermintrude",
            "password"        : "m00",
            "port"            : 3306,
            "failover_connections" :
                [
                  {
                      "host"  : "moogbox2",
                      "port"  : 3306
                  },
                  {
                      "host"  : "moogbox3",
                      "port"  : 3306
                  }
                ]
        },
```

On startup, the Cisco Crosswork Situation Manager components that make a MySQL connection will connect to the MySQL server on moogbox1.

If the MySQL server on `moogbox1` goes down then the Cisco Crosswork Situation Manager components will automatically failover their MySQL connection to `moog box2` next. If that is not available or subsequently goes down then the connection will failover to `moogbox3`. Whilst the failover is occurring, some temporary MySQL connection errors or warnings may be seen in the Cisco Crosswork Situation Manager components log output.

> If the primary or another `failover_connection` higher up the list becomes available again, the connection will not automatically failback to that until the Cisco Crosswork Situation Manager component is restarted or makes a new connection

HA - Deployment Scenarios

The architectures described here are for example only and the internal expertise within your organization must validate deployment architecture with reference to HA goals, organizational standards, and system configuration limitations before deployment. The following sections provide detailed descriptions of the following Cisco Crosswork Situation Manager deployment scenarios using High Availability system architecture:

- Single server with load balancer
- Three servers, two Clusters with load balancers
- Fully distributed multi-server

Considerations for Single and Multi-tiered Architecture

	Advantage	Disadvantage
Single-tier	maintenance	single point of failure
	budget considerations	CPU resource
	zero network latency between components	memory resource
	simple upgrade	security - single point of access
Multi-tier	db has dedicated machine - higher performance	possible network latency

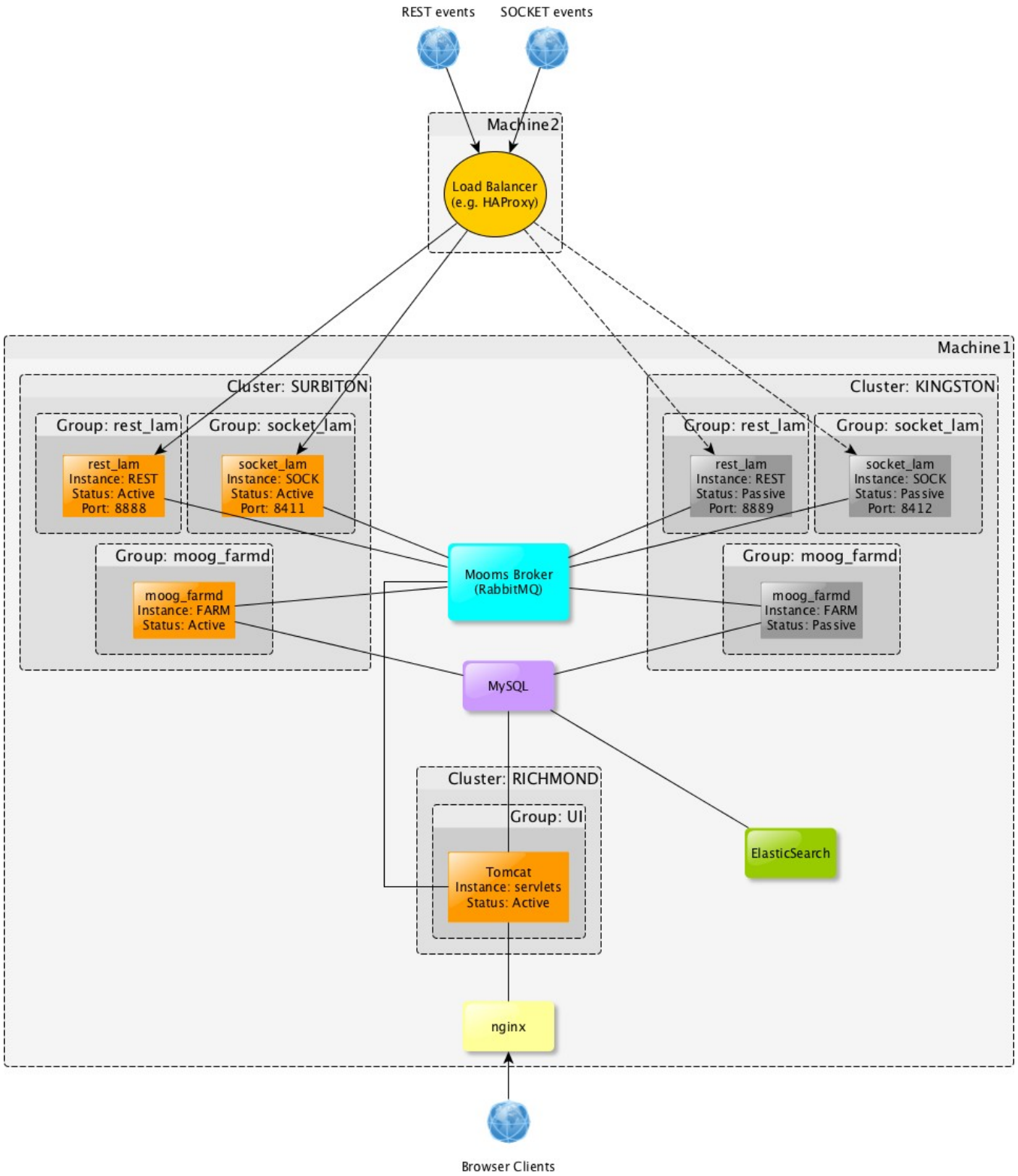
	greater performance - machine dependent	time consuming upgrade
	upgrade a node at a time	maintenance
	UI Clients performance increased	farmd - no horizontal processing
	LAMs performance increased	
	high availability	
	firewalls and security	

Single server setup with load balancer for testing/proving purposes

Whilst HA is aimed at distributed setups it may be useful to perform an "all-on-one-box" install for testing/proving purposes. Instances/Process Groups/Clusters can still be configured within this single server and failovers triggered using the ha_cntl utility.

> It is not possible to have multiple Instances of the Tomcat servlets on one machine so this system will be limited to the redundancy of the LAMs and moog_farmd

Architecture diagram



High-level description

- **Machine1** hosts:
  - Active Instances of the socket_lam, rest_lam, moog_farmd and Tomcat - all part of cluster 'SURBITON'
  - Passive Instances of the socket_lam, rest_lam, moog_farmd and Tomcat - all part of cluster 'KINGSTON'
  - MooMS broker (RabbitMQ)
  - MySQL server
  - ElasticSearch

    - nginx
- **Machine2** hosts:
    - LAM load balancer (e.g. HAProxy)

Purpose of system

This system provides all-in-one box redundancy for the REST LAM, the socket LAM and moog_farmd, so it can be used to test failover of those components. The UI is not part of that redundancy and is configured in its own separate Cluster to separate it logically from the other Clusters.

The load balancer is configured to:

- Route Events to the Active rest_lam, based on the listening port of the LAM being available and the `hastatus` endpoint not returning a `503 Service Unavailable`
- Route Events to the Active socket_lam, based on the listening port of the LAM being available and accepting connections. The LAM's configuration has `accept_conn_when_passive` set to `false` to ensure connection attempts are rejected when in Passive mode

Single-server full product installation

> All components installed at the same location

1. Run:

   ```
   yum groupinstall moogsoft
   ```

   or

   ```
   yum install moogsoft-db moogsoft-lams moogsoft-mooms moogsoft-search
   moogsoft-server moogsoft-ui moogsoft-utils
   ```

2. Run:

   ```
   $MOOGSOFT_HOME/bin/utils/moog_init.sh -I <ZONE> -u root
   ```

   where `<ZONE>` is the name of the MooMS zone you want to create.

3. Enter the password for MySQL. The `moog_init` script prompts for the MySQL root user password (blank by default) and then prompts for whether you wish to change the hostname used by the configurations (defaults to the machine hostname returned by the 'hostname' command).

Configuration

Component	Details
General	In file $MOOGSOFT_HOME/config/system.conf set the following properties (changing them from their defaults): • mooms.message_persistence : true • failover.persist_state : true • failover.hazelcast.hosts : ["<Machine1>"]
rest_lam	1. Create a copy of $MOOGSOFT_HOME/config/rest_lam.conf as $MOOGSOFT_HOME/config/rest_lam1.conf 1. Edit the ha section and set the following properties:

```
ha:
  {
    cluster:
"SURBITON",
    group:
"rest_lam",
    instance:
"REST",

start_as_passive:
false,

duplicate_source:
false
  },
```

2. Create another copy as $MOOGSOFT_HOME/config /rest_lam2.conf.
    1. Set a different "port" property to that of the first rest_lam e.g 8889
    2. Edit the ha section and set the following properties:

```
ha:
  {
    cluster:
"KINGSTON",
    group:
"rest_lam",
    instance:
"REST",

start_as_passive:
true,

duplicate_source:
false
  },
```

3. Create a copy of /etc/init.d/restlamd as /etc/init.d/restlamd1.
    1. Set the CONFIG_FILE property to point to rest_lam1.conf.
4. Create a copy of /etc/init.d/restlamd as /etc/init.d/restlamd2.
    1. Set the CONFIG_FILE property to point to rest_lam2.conf.
5. Start both services:

```
service restlamd1 start
service restlamd2 start
```

socket_lam	

1. Create a copy of $MOOGSOFT_HOME/config/socket_lam.conf as $MOOGSOFT_HOME/config/socket_lam1.conf.
    1. Edit the ha section and set the following properties:

```
ha:
  {
    cluster:
"SURBITON",
    group:
"socket_lam",
    instance:
"SOCK",

only_leader_active:
true,

accept_conn_when_pa
ssive: false,

start_as_passive:
false,

duplicate_source:
false
  },
```

2. Create another copy as $MOOGSOFT_HOME/config/socket_lam2.conf.
    1. Set a different "port" property to that of the first socket_lam e.g 8412
    2. Edit the ha section and set the following properties:

```
ha:
  {
    cluster:
"KINGSTON",
    group:
"socket_lam",
    instance:
"SOCK"

only_leader_active:
true,

accept_conn_when_pa
ssive: false,

start_as_passive:
true,

duplicate_source:
false
  },
```

	3. Create a copy of /etc/init.d/socketlamd as /etc/init.d /socketlamd1.
	1. Set the CONFIG_FILE property to point to socket_lam1.conf.
	4. Create a copy of /etc/init.d/socketlamd as /etc/init.d /socketlamd2.
	1. Set the CONFIG_FILE property to point to socket_lam2.conf.
	5. Start both services:
	`service socketlamd1 start`
	`service socketlamd2 start`

moog_farmd	1. Create a copy of $MOOGSOFT_HOME/config/moog_farmd.conf as $MOOGSOFT_HOME/config/moog_farmd1.conf.
	2. Edit $MOOGSOFT_HOME/config/moog_farmd1.conf and:
	1. Configure it with the required Moolets and **persist_state** settings for those moolets.
	2. Edit the ha section and set the following properties:
	`ha: { cluster: "SURBITON", group: "moog_farmd", instance: "FARM", start_as_passive: false },`
	3. Create a copy of $MOOGSOFT_HOME/config /moog_farmd1.conf as $MOOGSOFT_HOME/config /moog_farmd2.conf.
	4. Edit $MOOGSOFT_HOME/config/moog_farmd2.conf and:
	1. Configure it with the required Moolets and **persist_state** settings for those moolets.
	2. Edit the ha section and set the following properties:
	`ha: { cluster: "KINGSTON", group: "moog_farmd", instance: "FARM", start_as_passive: true },`

5. Create a copy of /etc/init.d/moogfarmd as /etc/init.d/moogfarmd1
    1. Set the CONFIG_FILE property to point to moog_farmd1.conf
6. Create a copy of /etc/init.d/moogfarmd as /etc/init.d/moogfarmd2
    1. Set the CONFIG_FILE property to point to moog_farmd2.conf
7. Start both services:

```
service moogfarmd1 start
service moogfarmd2 start
```

UI

1. Configure the /usr/share/moogsoft/config/servlets.conf file as follows:

```
{
   loglevel: "WARN",
   webhost :
"https://<Machine1>",
   moogsvr:
   {
        eula_per_user:
false,
        cache_root: "
/var/lib/moogsoft/moog-
data",

db_connections:
10,

priority_db_connections:
25
    },
    moogpoller :
    {
    },
    toolrunner :
    {
        sshtimeout:
900000,
        toolrunnerhost:
"<Machine1>",
        toolrunneruser:
"<toolrunner username>",

toolrunnerpassword:
"<toolrunner password>"
    },
    graze  :
    {
    },
```

```
    events  :
    {
    },
    ha :
    {
        cluster:
"RICHMOND",
        instance:
"servlets",
        group: "UI"

start_as_passive: false
    }
}
```

...replacing <hostname>, <toolrunner host>, <toolrunner username> and <toolrunner password> with appropriate values.

2. Restart the Apache-tomcat service with the following command:

```
service apache-tomcat
restart
```

Load Balancer

Example HAProxy configuration on **Machine2** for rest_lam and socket_lam:

```
global
  log 127.0.0.1   local0
  log 127.0.0.1   local1
notice
  maxconn 4096
  chroot /var/lib/haproxy
  user haproxy
  group haproxy
  daemon
  #debug
  #quiet

defaults
  mode tcp
  maxconn 10000
  timeout connect 5s
  timeout client 100s
  timeout server 100s

listen stats :9090
  balance
  mode http
  stats enable
  stats auth admin:admin
```

```
frontend rest_lam_frontend
  bind Machine2:8888
  mode http
  default_backend
rest_lam_backend

backend rest_lam_backend
  balance roundrobin
  mode http
  option httpchk POST
  http-check expect ! status
503
  server rest_lam_1 Machine1:
8888 check
  server rest_lam_2 Machine1:
8889 check

frontend socket_lam_frontend
  bind Machine2:8411
  mode tcp
  default_backend
socket_lam_backend

backend socket_lam_backend
  balance roundrobin
  mode tcp
  server socket_lam1
Machine1:8411 check
  server socket_lam2
Machine1:8412 check
```

-
- This config offers port 8888 for REST events and port 8411 for SOCKET events.
- http mode is used for the rest_lam and tcp mode for the socket_lam
  The httpchk option is used to get the Active/Passive status of the rest_lam and to treat a response of 503 (Passive) as if the LAM was down

Example failover commands

rest_lam failover	To failover only the rest_lam Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.rest_lam.REST` Instance and activating the `KINGSTON.rest_lam.REST` Instance) run	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.rest_lam`

socket_lam failover	To failover only the socket_lam Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.socket_lam.SOCK` Instance and activating the `KINGSTON.socket_lam.SOCK` Instance) run	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.socket_lam`
moog_farmd failover	To failover only the moog_farmd Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the SURBITON.moog_farmd.FARM Instance and activating the KINGSTON.moog_farmd.FARM Instance) run	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.moog_farmd`

Any of the above can be failed back individually by reactivating the Process Group in the SURBITON Cluster, for example:		`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON.rest_lam`
		`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON.socket_lam`
		`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON.moog_farmd`

It is possible to failover the LAMs and moog_farmd together from the SURBITON Cluster to the KINGSTON Cluster by activating the whole KINGSTON Cluster, for example:

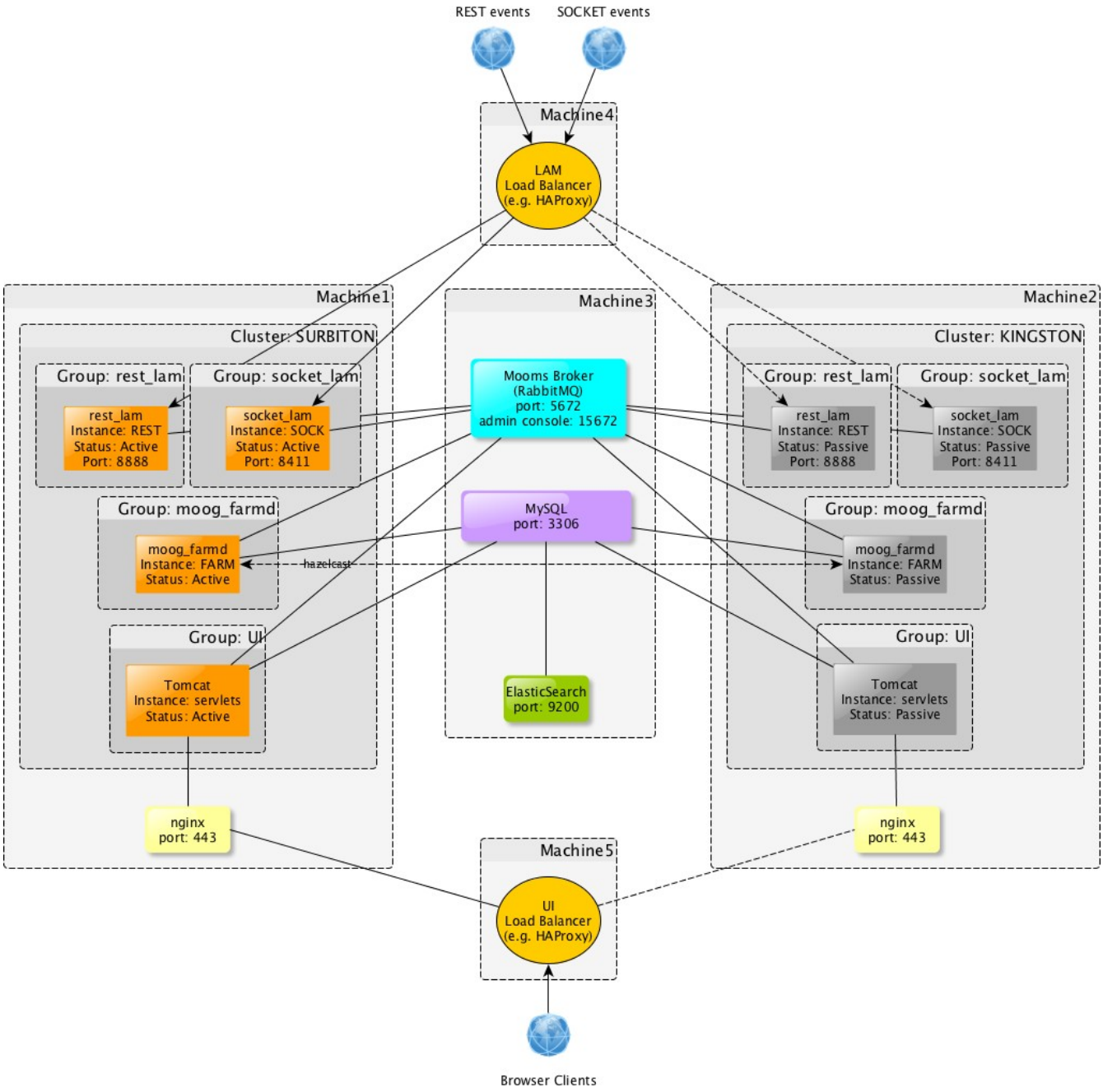
```
$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON
```

however this would have the side-effect of also deactivating the RICHMOND Cluster i.e. the UI servlets would all become Passive and the UI could not be used until all three servlets were reactivated

Three server, two Cluster setup with load balancers

This setup offers redundancy of Event ingestion, Event processing and UI components across two servers; each designated to be a Cluster. The MooMS broker and MySQL server are installed on a dedicated third server. Separate load balancer servers are also configured to route Events to the LAMs and web traffic to the UI Instances.

Architecture Diagram



High-level description

- **Machine1** hosts Active Instances of the socket_lam, rest_lam, moog_farmd and UI - all part of cluster 'SURBITON'
- **Machine2** hosts Passive Instances of the socket_lam, rest_lam, moog_farmd and UI - all part of cluster 'KINGSTON'
- **Machine3** hosts the following:
  - MooMS broker (RabbitMQ)
  - MySQL server
  - ElasticSearch
- **Machine4** hosts the LAM load balancer (e.g. HAProxy) which routes Events to only the Active Instances of both the rest_lam and socket_lam. In the event of a LAM failover, the load balancer switches the routing to the new Active Instances of those LAMs
- **Machine5** hosts the UI load balancer (e.g. HAProxy) which routes web traffic only to the nginx behind which is an active UI.

Purpose of system

This system provides redundancy for the REST LAM, the socket LAM, moog_farmd (using Hazelcast persistence) and the UI. The system makes use of a core server containing the MooMS broker, MySQL server and Elasticsearch server.
The LAM load balancer is configured to:

- Route Events to the Active rest_lam, based on the listening port of the LAM being available and the `hastatus` endpoint not returning a `503 Service Unavailable`
- Route Events to the Active socket_lam, based on the listening port of the LAM being available and accepting connections. The LAM's configuration has `accept_conn_when_passive` set to `false` to ensure connection attempts are rejected when in Passive mode

The UI load balancer is configured to:

- Route web traffic only to nginx behind which is an active UI. The decision for this is based on a check of the moogsvr hastatus endpoint in Tomcat.

Installation

Step		

1	***Install on Machine3***	1 Install the `db`, `mooms`, `search` and `utils` RPMs:
		`yum install moogsoft-db moogsoft-mooms moogsoft-search moogsoft-utils`
		2 Initialize the database:
		`$MOOGSOFT_HOME/bin/utils/moog_init_db.sh -Iu root`
		You are prompted for the password associated with the MySQL root user (blank by default).
		3 Connect to mysql as the root user and grant all access on the **moogdb** and **moog_reference** databases to the ermintrude user on both Machine1 and Machine2:
		`GRANT ALL ON moogdb.* TO ermintrude@'Machine1' IDENTIFIED BY 'm00';`

```
GRANT ALL
ON
moog_refer
ence.* TO
ermintrude
@'Machine1
'
IDENTIFIED
BY 'm00';
GRANT ALL
ON moogdb.
* TO
ermintrude
@'Machine2
'
IDENTIFIED
BY 'm00';
GRANT ALL
ON
moog_refer
ence.* TO
ermintrude
@'Machine2
'
IDENTIFIED
BY 'm00';
```

4. Initialize the MooMS message broker (by creating a 'zone' and enabling the management plugins):

```
$MOOGSOFT_
HOME/bin
/utils
/moog_init
_mooms.sh
-pz <ZONE>
```

where `<ZONE>` is the name of the MooMS zone you want to create.

5. Configure search to connect to MySQL and set up the indexer `cron` job

```
$MOOGSOFT_
HOME/bin
/utils
/moog_init
_search.
sh -sd
localhost:
3306
```

		6. Configure the **ElasticSearch** process to listen on all interfaces (so can accept remote connections) by adding the following line to the /etc/elasticsearch/elasticsearch.yml file and then restarting the elasticsearch service:
		`http.host: 0.0.0.0`
		7. Configure the utils with connection details for MySQL and the MooMS broker:
		`$MOOGSOFT_HOME/bin/utils/moog_init_utils.sh -z <ZONE> -d Machine3:3306 -m Machine3:5672`
		Ensure that the `<ZONE>` used here is the same as used in step 3 above when initializing the MooMS message broker

2	Install on Machine1	1 Install the `lams`, `server`, `ui` and `utils` RPMs: `yum install moogsoft-lams moogsoft-server moogsoft-ui moogsoft-utils` If you encounter dependency errors

involving MySQL, you may have older versions of MySQL libraries on your server. These prevent installation of the Cisco Crosswork Situation Manager components and you should therefore take care should not to simply remove these outright in case other dependent packages are impacted. Best practice is to create a script that is used by the `yum shell` command.
For example, if an existing `mysql-libs` prevented Cisco Crosswork Situation Manager installati on, then create a file `/tmp/install` with the following contents:

```
remove mysql-libs
install moogsoft-lams
install moogsoft-server
install moogsoft-ui
install moogsoft-utils
run
```

and the script fed to `yum shell` using:

```
cat
/tmp
/ins
tall
|yum
shell
```

This displays what is going to be installed and what is going to be removed. If this looks correct, then re-run it with the `-y` flag to carry out the installation, as follows:

```
cat
/tmp
/ins
tall
|yum
shel
l -y
```

this progresses the installation, removing and replacing the erroneous package without causing dependency errors.

2 Configure the LAMs to connect to the database and MooMS broker on **Machine3**:

```
$MOOGSOFT_
HOME/bin
/utils
/moog_init
_lams.sh -
bz <ZONE>
-d
Machine3:
3306 -m
Machine3:
5672
```

Ensure that the `<ZONE>` used here is the same as used in Part 1 step 3 above, when initializing the MooMS message broker

The `-b` option backs up the original `config` and `bot` files.

3 Configure the events analyzer `cron` job:

```
$MOOGSOFT_
HOME/bin
/utils
/moog_init
_server.
sh -e
```

> The `moog_init_serve r.sh` script can also configure the `system. conf` database and MooMS settings. However, as this has already been done in the previous step, there is no need to repeat it here

4 Configure the UI:

```
$MOOGSOFT_
HOME/bin
/utils
/moog_init
_ui.sh -
otwfxz
<ZONE> -c
Machine3:
15672 -s
Machine3:
9200
```

> Ensure that the `<ZONE>` used here is the same as used in Part 1 step 3 above, when initialising the MooMS message broker

This does the following:

- Configures the UI with a connection to the Elasticsearch server running on `Mac hine3: 9200`
- sets the MooMS console to `Machine3: 15672`
- sets the UI zone to `<ZONE>`
- Generates `ssl` keys for Nginx
- Rebuilds the webapps
- Restarts the Tomcat service

		/There is no need to run `moog_init_utils.sh` at this stage as everything it requires has been configured.
3	***Install on Machine2***	This is identical to that of **Machine1** so repeat the steps in Part 2 above on **Machine2**.

Configuration

Cisco Crosswork Situation Manager uses Hazelcast as a persistence mechanism.

Component	Details
General	On both **Machine1** and **Machine2** edit file $MOOGSOFT_HOME/config/system.conf and set the following properties (changing them from their defaults): • mooms.message_persistence : true • failover.persist_state : true • failover.hazelcast.hosts : ["<Machine1>","<Machine2>"]
rest_lam	1. On **Machine1** for the **active** rest_lam edit file $MOOGSOFT_HOME/config/rest_lam.conf and uncomment/edit the ha section as follows: `ha: { cluster: "SURBITON", group: "rest_lam", instance: "REST", start_as_passive: false, duplicate_source: false },`

	2. On **Machine2** for the **passive** rest_lam edit file $MOOGSOFT_HOME/config /rest_lam.conf and uncomment/edit the ha section as follows:
	`ha: { cluster: "KINGSTON" , group: "rest_lam" , instance: "REST", start_as_passive: true, duplicate_source: false },`
	3. Start the **restlamd** services on both machines:
	`service restlamd start`

socket_lam	1. On **Machine1** for the **active** socket_lam edit file $MOOGSOFT_HOME/config /socket_lam.conf and uncomment /edit the ha section as follows: `ha: { cluster: "SURBITON" , group: "socket_lam",`

```
instance:
"SOCK",

only_leade
r_active:
true,

accept_con
n_when_pas
sive:
false,

start_as_p
assive:
false,

duplicate_
source:
false
  },
```

2. On **Machine2** for the **passive** socket_lam edit file $MOOGSOFT_HOME/config /socket_lam.conf and uncomment /edit the ha section as follows:

```
ha:
  {

cluster:
"KINGSTON"
,

group:
"socket_la
m",

instance:
"SOCK",

only_leade
r_active:
true,

accept_con
n_when_pas
sive:
false,

start_as_p
assive:
true,
```

	<pre>duplicate_
	source:
	false
	},</pre>
	3. Start the **socketlamd** services on both machines:	

moog_farmd	1. Edit the $MOOGSOFT_HOME /config/moog_farmd.conf file on both **Machine1** and **Machine2** and configure it with the required Moolets and **persist_state** settings for those moolets.
	
	Other than the **ha** section, all settings in moog_farmd.conf must be identically configured on both **Machine1** and **M achine2**
	
	2. On **Machine1** edit the $MOOGSOFT_HOME/config /moog_farmd.conf file and set the ha section as follows:
	
	<pre>ha:
	{
	
	cluster:
	"SURBITON"
	,
	
	group:
	"moog_farm
	d",
	
	instance:
	"FARM",
	
	start_as_p
	assive:
	false
	},</pre>
	3. On **Machine2** edit the $MOOGSOFT_HOME/config /moog_farmd.conf file and set the ha section as follows:
	
	<pre>ha:
	{</pre>	

```
cluster:
"KINGSTON"
,

group:
"moog_farm
d",

instance:
"FARM",

start_as_p
assive:
true
  },
```

4. Start the services on both machines.

```
service
moogfarmd
start
```

UI

1. On **Machine1** edit the $MOOGSOFT_HOME/config /servlets.conf file as follows:

```
{

loglevel:
"WARN",

webhost :
"https://<
Machine5>"
,

moogsvr:
   {

eula_per_u
ser:
false,

cache_root
: "/var
/lib
/moogsoft
/moog-
data",
```

```
db_connect
ions:
10,

priority_d
b_connecti
ons: 25
    },

moogpoller
:
    {
    },

toolrunner
:
    {

sshtimeout
: 900000,

toolrunner
host:
"<Machine1
>",

toolrunner
user:
"<toolrunn
er
username>"
,

toolrunner
password:
"<toolrunn
er
password>"
    },

graze  :
    {
    },

events  :
    {
    },
    ha :
    {

cluster:
"SURBITON"
,
```

```
instance:
"servlets"
,

group:
"UI",

start_as_p
assive:
false
    }
}
```

2. On **Machine2** edit the $MOOGSOFT_HOME/config /servlets.conf file as follows:

```
{

loglevel:
"WARN",

webhost :
"https://<
Machine5>"
,

moogsvr:
   {

eula_per_u
ser:
false,

cache_root
: "/var
/lib
/moogsoft
/moog-
data",

db_connect
ions:
10,

priority_d
b_connecti
ons: 25
    },

moogpoller
```

```
:
    {
    },

toolrunner
:
    {

sshtimeout
: 900000,

toolrunner
host:
"<Machine2
>",

toolrunner
user:
"<toolrunn
er
username>"
,

toolrunner
password:
"<toolrunn
er
password>"
    },

graze  :
    {
    },

events  :
    {
    },
    ha :
    {

cluster:
"KINGSTON"
,

instance:
"servlets"
,

group:
"UI",

start_as_p
assive:
```

	<pre>true
	}
	}</pre>
	3. Restart the Apache-tomcat service on both **Machine1** and **Machine2**:
	<pre>service
	apache-
	tomcat
	restart</pre>

LAM Load Balancer	Example HAProxy configuration on **Machine4** for rest_lam and socket_lam:
	<pre>global
	log
	127.0.0.1
	local0
	log
	127.0.0.1
	local1 notice
	maxconn 4096
	chroot /var
	/lib/haproxy
	user haproxy
	group haproxy
	daemon
	#debug
	
	defaults
	mode tcp
	maxconn 10000
	timeout
	connect 5s
	timeout
	client 100s
	timeout
	server 100s
	
	listen stats :
	9090
	balance
	mode http
	stats enable
	stats auth
	admin:admin
	
	frontend
	rest lam fronte</pre>

```
nd
  bind
Machine4:8888
  mode http

default_backend
rest_lam_backen
d

backend
rest_lam_backen
d
  balance
roundrobin
  mode http
  option
httpchk POST
  http-check
expect !
status 503
  server
rest_lam_1
Machine1:8888
check
  server
rest_lam_2
Machine2:8888
check

frontend
socket_lam_fron
tend
  bind
Machine4:8411
  mode tcp

default_backend
socket_lam_back
end

backend
socket_lam_back
end
  balance
roundrobin
  mode tcp
  server
socket_lam1
Machine1:8411
check
  server
socket_lam2
Machine2:8411
```

```
  check
```

- This config offers port 8888 for REST events and port 8411 for SOCKET events.
- `http` mode is used for the rest_lam and `tcp` mode for the socket_lam
- The `httpchk` option is used to get the Active/Passive status of the rest_lam and to treat a response of `503` (Passive) as if the LAM was down

UI Load Balancer

Example HAProxy configuration on **Machine5** for the UI:

```
global
  log
127.0.0.1
local0
  log
127.0.0.1
local1 notice
  maxconn 4096
  chroot /var
/lib/haproxy
  user haproxy
  group haproxy
  daemon
  #debug

defaults
  mode tcp
  maxconn 10000
  timeout
connect 5s
  timeout
client 24d
  timeout

listen stats :
9090
  balance
  mode http
  stats enable
  stats auth
admin:admin

frontend
```

```
ui_front
  bind
Machine5:443
  option
tcplog
  mode tcp

default_backend
ui back

backend
ui_back
  mode tcp
  balance
source
  option
httpchk GET
/moogsvr
/hastatus
  http-check
expect status
204
  server ui_1
Machine1:443
check check-
ssl verify
none inter 100
  server ui_2
Machine2:443
check check-
ssl verify
```

- This config offers port **443** for UI traffic.
- **tcp** mode is used throughout to handle the https traffic in **ssl** passthrough mode
- Note the setting of "timeout client" and "timeout server" to the maximum allowable of 24d (days) - this ensures the browser-server websocket connection does not get disconnected.
- The **httpchk** option is used against the /moogsvr/hastatus endpoints to get their Active /Passive status (a 204 response means active, a 503 response is passive)
- **Source** balancing is used but as there is only one active UI server at a time the balancing mechanism is irrelevant.

Example failover commands

Command	Description	Command line
rest_lam failover	To failover only the rest_lam Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.rest_lam.REST` Instance and activating the `KINGSTON.rest_lam.REST` Instance)	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.rest_lam`
socket_lam failover	To failover only the socket_lam Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.rest_lam.SOCK` Instance and activating the `KINGSTON.rest_lam.SOCK` Instance)	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.socket_lam`
moog_farmd failover	To failover only the moog_farmd Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.moog_farmd.FARM` Instance and activating the KINGSTON.moog_farmd.FARM Instance)	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.moog_farmd`

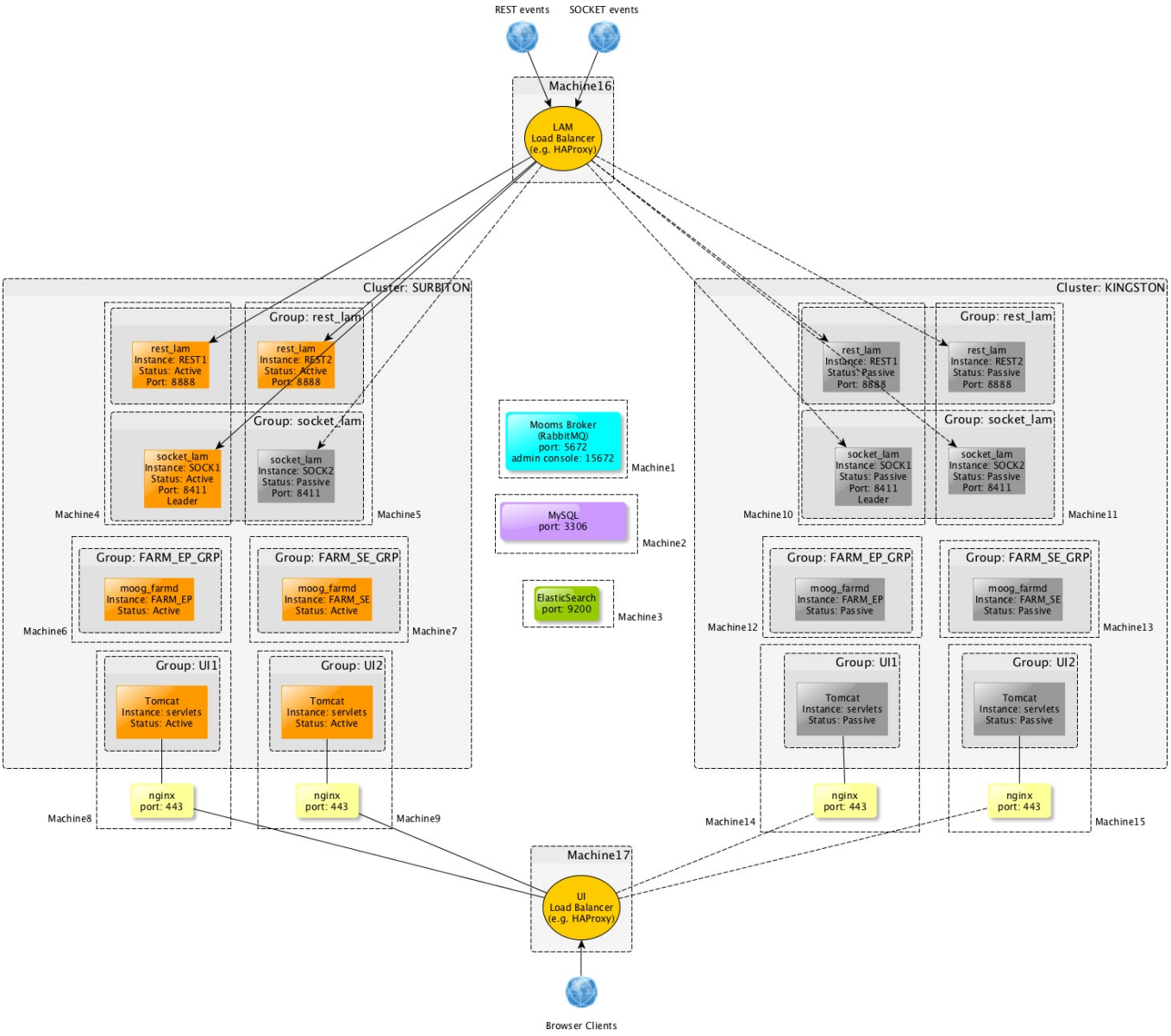
Any of the above can be failed back individually by reactivating the Process Group in the SURBITON Cluster, for example:		`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON.rest_lam`
		`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON.socket_lam`
		`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON.moog_farmd`

Cluster failover	To failover the entire SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating everything in the SURBITON Cluster and activating everything in the KINGSTON Cluster)	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON`
	To fail back to the SURBITON Cluster again	`$MOOGSOFT_HOME/bin/ha_cntl -a SURBITON`

Fully distributed multi-server setup

This setup offers redundancy of multiple components across multiple servers with two Clusters. The MooMS broker and MySQL server and Search (Elasticsearch) components are each installed on their own dedicated servers. Separate load balancer servers are also configured to route Events to the LAMs and web traffic to the UI Instances.

Architecture Diagram



> For clarity, this diagram does not show all inter-component connection lines

High-level description

- **Machine1** hosts the MooMS broker

- **Machine2** hosts the MySQL database
- **Machine3** hosts ElasticSearch
- **Machine4** hosts the active REST1 and active SOCK1 instances of the REST and SOCKET lams respectively - all part of the SURBITON cluster.
- **Machine5** hosts the active REST2 and passive SOCK2 instances of the REST and SOCKET lams respectively - all part of the SURBITON cluster.
- **Machine6** hosts the active FARM_EP instance of moog_farmd, part of the SURBITON cluster.
- **Machine7** hosts the active FARM_SE instance of moog_farmd, part of the SURBITON cluster.
- **Machine8** hosts nginx and active UI group UI1, part of the SURBITON cluster.
- **Machine9** hosts nginx and active UI group UI2, part of the SURBITON cluster.
- **Machine10** hosts the passive REST1 and passive SOCK1 instances of the REST and SOCKET lams respectively - all part of the KINGSTON cluster.
- **Machine11** hosts the passive REST2 and passive SOCK2 instances of the REST and SOCKET lams respectively - all part of the KINGSTON cluster.
- **Machine12** hosts the passive FARM_EP instance of moog_farmd, part of the KINGSTON cluster.
- **Machine13** hosts the passive FARM_SE instance of moog_farmd, part of the KINGSTON cluster.
- **Machine14** hosts nginx and passive UI group UI1, part of the KINGSTON cluster.
- **Machine15** hosts nginx and passive UI group UI2, part of the KINGSTON cluster.
- **Machine16** hosts the LAM load balancer (e.g. HAProxy) which routes Events to only the Active Instances of both the rest_lam and socket_lam. In the event of a LAM failover, the load balancer switches the routing to the new Active Instances of those LAMs
- **Machine17** hosts the UI load balancer (e.g. HAProxy) which routes web traffic only to an nginx behind which is an active UI.

Purpose of system

This system provides:

- Redundancy for the rest_lam, the socket LAM, moog_farmd and the UI
- Load balancing capability for the rest_lam and UI within a single Cluster
- The ability to failover socket_lam to another machine within a Cluster
- Distributed moog_farmd processing

The system makes use of three core servers containing the MooMS broker, MySQL server and Elasticsearch. The MooMS broker and MySQL server here could be replaced with clustered versions of each.
The LAM load balancer is configured to:

- Round-robin Events between the pair of Active rest_lams in the Cluster and in the event of a failover, route Events to the newly Active rest_lam(s) in the other Cluster
- Route events to the Active socket_lam and in the event of a failover, route Events to the newly Active socket_lam(s)

The UI load balancer is configured to:

- Round-robin https traffic (ideally based on sticky sessions or "source") between the pair of Active UI groups and in the event of a failover route traffic to the newly active UI groups in the other Cluster

Installation

Step	*Machine*	Description	Command line
1. Install the MooMS component	1	Install the `mooms` RPM	`yum install moogsoft-mooms`
		Initialize the MooMS message broker (by creating a 'zone' and enabling the management plugins) Where `<ZONE>` is the name of the MooMS zone you want to create	`$MOOGSOFT_HOME/bin/utils/moog_init_mooms.sh -pz <ZONE>`

2. Install and configure the database component	2	Install the `db` RPM	`yum install moogsoft-db`
		Initialize the database You are prompted for the password associated with the MySQL root user (blank by default)	`$MOOGSOFT_HOME/bin/utils/moog_init_db.sh -Iu root`
		Connect to mysql as the root user and grant all access on the **moogdb** and **moog_reference** databases to the ermintrude user from any machine.	`GRANT ALL ON moogdb.* TO ermintrude@'%' IDENTIFIED BY 'm00';` `GRANT ALL ON moog_reference.* TO ermintrude@'%' IDENTIFIED BY 'm00';`
3. Install and configure the search component	3	Install the `search` RPM	`yum install moogsoft-search`
		Configure search to connect to MySQL and set up the indexer `cron` job	`$MOOGSOFT_HOME/bin/utils/moog_ini`

			`t_search.sh -sd Machine2:3306`
		Configure the **ElasticSearch** process to listen on all interfaces (so can accept remote connections) by adding the following line to the /etc/elasticsearch/elasticsearch.yml file and then restart the elasticsearch service.	`http.host: 0.0.0.0`
4. Install the LAMs components	4 5 10 11	Install the `lams` RPM on **each** machine	`yum install moogsoft-lams`
		Configure the LAMs to connect to the database on **Machine2** and the MooMS broker on **Machine1** Ensure that the `<ZONE>` used here is the same as used in Part 1 step 2 above, when initializing the MooMS message broker The `-b` option backs up the original `config` and `bot` files	`$MOOGSOFT_HOME/bin/utils/moog_init_lams.sh -bz <ZONE> -d Machine2:3306 -m Machine1:5672`
5. Install the server and utils components	6 7 12 13	Install the `server` and `utils` RPMs on **each** machine	`yum install moogsoft-server moogsoft-utils`
		Configure the server components to connect to the database on **Machine2** and the MooMS broker on **Machine1**, configure the events analyzer `cron` job Ensure that the `<ZONE>` used here is the same as used in Part 1 step 2 above, when initializing	`$MOOGSOFT_HOME/bin/utils/moog_init_server.sh -bez <ZONE> -d`

		the MooMS message broker The `-b` option backs up the original `config` and `bot` files There is no need to run `moog_init_utils.sh` at this stage as everything it needs has already been configured	`Machine2:3306 -m Machine1:5672`
6. Install the UI component	8 9 14 15	Install the `ui` RPM on **each** machine	`yum install moogsoft-ui`
		Configure the UI component to connect to the database on **Machine2**, the MooMS broker on **Machine1**, and Elasticsearch on **Machine3** Ensure that the `<ZONE>` used here is the same as used in Part 1 step 2 above, when initializing the MooMS message broker	`$MOOGSOFT_HOME/bin/utils/moog_init_ui.sh -otwfxz <ZONE> -c Machine1:15672 -d Machine2:3306 -m Machine1:5672 -s Machine3:9200`

Configuration

Component	Details
rest_lam	Using the same mechanisms as described for the rest_lam in the previous 2 system examples, carry out the following actions: **Machine4**: Configure ha section in rest_lam.conf with instance: REST1, group: rest_lam, cluster: SURBITON and start_as_passive: false **Machine5**: Configure ha section in rest_lam.conf with instance: REST2, group: rest_lam, cluster: SURBITON and start_as_passive: false **Machine10**: Configure ha section in rest_lam.conf with instance: REST1, group: rest_lam, cluster: KINGSTON and start_as_passive: true

	Machine11: Configure ha section in rest_lam.conf with instance: REST1, group: rest_lam, cluster: KINGSTON and start_as_passive: true Start the **restlamd** service on all 4 machines.
socket_lam	Using the same mechanisms as described for the socket_lam in the previous 2 system examples, carry out the following actions: **Machine4**: Configure ha section in socket_lam.conf with instance: SOCK1, group: socket_lam, cluster: SURBITON, start_as_passive: false and accept_conn_when_passive: false **Machine5**: Configure ha section in socket_lam.conf with instance: SOCK2, group: socket_lam, cluster: SURBITON, start_as_passive: true, accept_conn_when_passive: false and default_leader: false **Machine10**: Configure ha section in socket_lam.conf with instance: SOCK1, group: socket_lam, cluster: KINGSTON, start_as_passive: true and accept_conn_when_passive: false **Machine11**: Configure ha section in socket_lam.conf with instance: SOCK2, group: socket_lam, cluster: KINGSTON, start_as_passive: true, accept_conn_when_passive: false and default_leader: false Start the **socketlamd** service on all 4 machines.
moog_farmd	Using the same mechanisms as described for moog_farmd in the previous 2 system examples, carry out the following actions: **Machine6:** • Configure system.conf and set mooms.message_persistence: true, failover.persist_state: true and failover.hazelcast.hosts: ["<Machine6>","<Machine12">] • Configure moog_farmd.conf with the required moolets (e.g. AlertBuilder, AlertRulesEngine, Sigaliser(s) & SituationMgr) and persist_state settings. • Configure ha section of moog_farmd with instance: FARM_EP, group: FARM_EP_GRP and cluster: SURBITON **Machine7:** • Configure system.conf and set mooms.message_persistence: true, failover.persist_state: true and failover.hazelcast.hosts: ["<Machine7>","<Machine13">] • Configure moog_farmd.conf with the required moolets (e.g. EmptyMoolet, ServiceNowmoolet) and persist_state settings. • Configure ha section of moog_farmd with instance: FARM_SE, group: FARM_SE_GRP and cluster: SURBITON **Machine12**: • Configure system.conf as for **Machine6** above. • Configure moog_farmd.conf with the required moolets as for **Machine6** above**.** • Configure ha section of moog_farmd with instance: FARM_EP, group: FARM_EP_GRP, cluster: KINGSTON and start_as_passive: true **Machine13:** • Configure system.conf as for **Machine7** above. • Configure moog_farmd.conf with the required moolets as for **Machine7** above**.** • Configure ha section of moog_farmd with instance: FARM_SE, group: FARM_SE_GRP, cluster: KINGSTON and start_as_passive: true Finally, start the **moogfarmd** services on **Machine6**, **Machine7**, **Machine12 and Machine13**
UI servlets	On **Machine8**, **Machine9**, **Machine14** and **Machine15**, configure system.conf to set mooms.message_persistence: true

	Using the same mechanisms as described for UI servlets in the previous 2 system examples, carry out the following actions: **Machine8:** Configure servlets.conf and set webhost : https://<Machine17> and in the ha section set instance: servlets, group: UI1, cluster: SURBITON and start_as_passive: false **Machine9:** Configure servlets.conf and set webhost : https://<Machine17> and in the ha section set instance: servlets, group: UI2, cluster: SURBITON and start_as_passive: false **Machine14:** Configure servlets.conf and set webhost : https://<Machine17> and in the ha section set instance: servlets, group: UI1, cluster: KINGSTON and start_as_passive: true **Machine15:** Configure servlets.conf and set webhost : https://<Machine17> and in the ha section set instance: servlets, group: UI2, cluster: KINGSTON and start_as_passive: true Finally, restart the **apache-tomcat** services on **Machine8**, **Machine9** , **Machine14** and **Machine15**:

LAM Load Balancer	The following is an example HAProxy configuration on **Machine16** for rest_lam and socket_lam.
	
	`global`
	`  log 127.0.0.1  local0`
	`  log 127.0.0.1  local1`
	`notice`
	`  maxconn 4096`
	`  chroot /var/lib/haproxy`
	`  user haproxy`
	`  group haproxy`
	`  daemon`
	`  #debug`
	`  #quiet`
	
	`defaults`
	`  mode tcp`
	`  maxconn 10000`
	`  timeout connect 5s`
	`  timeout client 100s`
	`  timeout server 100s`
	
	`listen stats :9090`
	`  balance`
	`  mode http`
	`  stats enable`
	`  stats auth admin:admin`
	
	`frontend rest_lam_frontend`
	`  bind Machine16:8888`
	`  mode http`
	`  default_backend`
	`rest_lam_backend`
	
	`backend rest_lam_backend`
	`  balance roundrobin`
	`  mode http`

```
  option httpchk POST
  http-check expect ! status
503
  server rest_lam_1 Machine4:
8888 check
  server rest_lam_2 Machine5:
8888 check
  server rest_lam_3
Machine10:8888 check
  server rest_lam_4
```

```
frontend socket_lam_frontend
  bind Machine16:8411
  mode tcp
  default_backend
socket_lam_backend

backend socket_lam_backend
  balance roundrobin
  mode tcp
  server socket_lam1
Machine4:8411 check
  server socket_lam2
Machine5:8411 check
  server socket_lam3
Machine10:8411 check
  server socket_lam4
```

- This config offers port 8888 for REST events and port 8411 for SOCKET events.
- http mode is used for the rest_lam and tcp mode for the socket_lam
- The httpchk option is used to get the Active/Passive status of the rest_lam and to treat a response of 503 (Passive) as if the LAM was down

UI Load Balancer

The following is an example HAProxy configuration on **Machine17** for the UI:

```
global
  log 127.0.0.1   local0
  log 127.0.0.1   local1
notice
  maxconn 4096
  chroot /var/lib/haproxy
  user haproxy
  group haproxy
  daemon
  #debug
```

```
  #quiet

defaults
  mode tcp
  maxconn 10000
  timeout connect 5s
  timeout client 24d
  timeout server 24d

listen stats :9090
  balance
  mode http
  stats enable
  stats auth admin:admin

frontend ui_front
  bind Machine17:443
  option tcplog
  mode tcp
  default_backend ui_back

 backend ui_back
  mode tcp
  balance source
  option httpchk GET /moogsvr
/hastatus
  http-check expect status
204
  server ui_1 Machine8:443
check check-ssl verify none
inter 50
  server ui_2 Machine9:443
check check-ssl verify none
inter 50
  server ui_3 Machine14:443
check check-ssl verify none
inter 50
  server ui_4 Machine15:443
check check-ssl verify none
```

- This config offers port **443** for UI traffic.
- **tcp** mode is used throughout to handle the https traffic in **ssl** passthrough mode
- Note the setting of "timeout client" and "timeout server" to the maximum allowable of 24d (days) - this ensures the browser-server websocket connection does not get disconnected.
- The **httpchk** option is used against the /moogsvr /hastatus endpoints to get their Active/Passive status (a 204 response means active, a 503 response is passive)
- **Source** balancing is used so that requests from different sources are always routed to the same backend server. If all browser clients are behind a NAT and present the same IP then this would not be a

	suitable method. Session stickiness is another configuration option - see this link: http://cbonte.github.io/haproxy-dconv/configuration-1.6.html#4-balance

Example failover commands

Command	Description	Command line
rest_lam failover	To failover only the rest_lam Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.rest_lam.REST1` and `SURBITON.rest_lam.REST2` Instances and activating the `KINGSTON.rest_lam.REST1` and `KINGSTON.rest_lam.REST2` Instances) run:	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.rest_lam`
	To failover only the `SURBITON.rest_lam.REST1` rest_lam Instance to the KINGSTON Cluster, this is done manually in two steps:	`$MOOGSOFT_HOME/bin/ha_cntl -d SURBITON.rest_lam.REST1` `$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.rest_lam.REST1`
socket_lam failover	To failover only the socket_lam Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.socket_lam.SOCK1` Instance and activating only the `KINGSTON.socket_lam.SOCK1` Instance - hence the "leader" setting) run:	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.socket_lam`
moog_farmd failover	To failover only the FARM_EP_GRP moog_farmd Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.moog_farmd.FARM_EP` Instance and activating the `KINGSTON.moog_farmd.FARM_EP` Instance) run:	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.FARM_EP`
	To failover only the FARM_SE_GRP moog_farmd Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the `SURBITON.moog_farmd.FARM_SE` Instance and activating the `KINGSTON.moog_farmd.FARM_SE` Instance) run:	`$MOOGSOFT_HOME/bin/ha_cntl -a KINGSTON.FARM_SE`
UI servlet failover	To failover only the UI1 Process Group from the SURBITON Cluster to the KINGSTON	

	Cluster (i.e. deactivating the SURBITON.UI1. servlets instance and activating the KINGSTON.UI1.servlets instance) run:	`$MOOGSOFT_HOME /bin/ha_cntl -a KINGSTON.UI1`
	To failover only the UI2 Process Group from the SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating the SURBITON.UI2. servlets instance and activating the KINGSTON.UI2.servlets instance) run:	`$MOOGSOFT_HOME /bin/ha_cntl -a KINGSTON.UI2`

	Any of the above can be failed back individually by reactivating the Process Group in the SURBITON Cluster, for example:	`$MOOGSOFT_HOME /bin/ha_cntl -a SURBITON. rest_lam`
		`$MOOGSOFT_HOME /bin/ha_cntl -a SURBITON. socket_lam`
		`$MOOGSOFT_HOME /bin/ha_cntl -a SURBITON. moog_farmd_EP`
		`$MOOGSOFT_HOME /bin/ha_cntl -a SURBITON. moog_farmd_SE`
		`$MOOGSOFT_HOME /bin/ha_cntl -a SURBITON.UI1`
		`$MOOGSOFT_HOME /bin/ha_cntl -a SURBITON.UI2`

Cluster failover	To failover the entire SURBITON Cluster to the KINGSTON Cluster (i.e. deactivating everything in the SURBITON Cluster and activating everything in the KINGSTON Cluster except the SOCK2 Instance as per the leader setting) run:	`$MOOGSOFT_HOME /bin/ha_cntl -a KINGSTON`
	To fail back to the SURBITON Cluster again, run:	`$MOOGSOFT_HOME /bin/ha_cntl -a SURBITON`

HA - Reference Architectures

Introduction

> This document describes various practical deployment scenarios. Readers are encouraged to first refer to the High Availability to understand HA fundamentals

The reference architectures described here are for example only and the internal expertise within your organization must validate deployment architecture with reference to HA goals, organizational standards, and system configuration limitations before deployment. The following HA reference architectures are included:

- Two Server Active/Passive
- Two Server Active/Active
- Four Server Active/Passive
- Four Server Active/Active
- Eight Server Active/Active Fully Distributed

Production implementations can use additional servers and combinations to address Scaling and Performance. These reference architectures are focused on showing HA configuration and failover scenarios. Scaling and Performance are not the focus. Also, these reference architectures are intended to complement the deployment scenarios section.

> - The reference architectures listed are based only on hazelcast as the mechanism for state persistence in a HA setup. Previously, both memcached and hazelcast mechanisms were available, but memcached has been removed as of 5.1.3 and later
> - Currently, failover of Cisco Crosswork Situation Manager components is **manually triggered** (no auto failover) using the ha_cntl command line utility

Considerations for Single and Multi-tiered Architecture

	Advantage	Disadvantage
Single-tier	maintenance	single point of failure
	budget considerations	CPU resource
	zero network latency between components	memory resource
	simple upgrade	security - single point of access
Multi-tier	db has dedicated machine - higher performance	possible network latency
	greater performance - machine dependent	time consuming upgrade
	upgrade a node at a time	maintenance
	UI Clients performance increased	farmd - no horizontal processing
	LAMs performance increased	firewalls and security
	high availability	

HA Feature Overview

In Cisco Crosswork Situation Manager, two or more servers can be clustered to provide redundancy. Clustered moog processes can be set to active or passive states. Processing can be moved from one server to another via the ha_cntl command. The ha_cntl command allows Cisco Crosswork Situation Manager clusters, process groups, or single process processing to be moved from one server to another, as well as add additional active processes (tomcat, LAMs) on one or more servers. Only one moogfarmd process can be active at a time because moogfarmd runs the Sigaliser process which create the Situations. The Sigaliser in-memory data is synced between moogfarmd processes with hazelcast. When switching over, the primary processes in the activated cluster will become active and the corresponding processes in the secondary cluster will be become inactive/passive.

Common Elements

User connections	run through a Server Load Balancer (SLB) that routes them to the active tomcat servlets
Tomcat	connects to MySQL during initial load then listens to the MooMS (RabbitMQ) bus for Alert and Situation updates. In Cisco Crosswork Situation Manager more than one MySQL and RabbitMQ servers can be specified
MySQL	is the main store/DB for Alert information. More than one MySQL server can be configured in the system.conf file allowing Cisco Crosswork Situation Manager processes to seek out the active MySQL host if the first one is down
moogfarmd	runs the software processing layer or the moolet processes (AlertBulder, Alert Rules Engine, Template Matcher, Sigalisers, Situation Manager, Notifier and other user moolets). The active moogfarmd synchronizes the passive moogfarmd via hazelcast. This allows a passive moogfarmd to become active and take up where the formerly active moogfarmd stopped processing. Most moolets have extensions called moobots which are JavaScript processing rules. The ones that do not are the Sigaliser (classic), SpeedBird, Template Matcher, and Nexus
RabbitMQ	is a clustered messaging bus. All process communication about events, Alerts and Situations use this bus. Tomcat, moogfarmd, and the LAMs subscribe and publish on the bus. It is also responsible for archiving events, Alerts, and Situations in the MySQL database
Link Access Modules (LAM)	processes raw feeds into events and publish them on the RabbitMQ bus. There are extensions on the LAM code that are similar to moobots called lambots. These like the moobots are also written in JavaScript Events entering the system are routed to the active LAMs via the Server Load Balancer (SLB)

Database

> Comprehensive discussion/detail about DB replication techniques/reference architectures is out of scope of this document. Most enterprises have a dedicated team that typically handle DB administration using their standard methods and procedures (Clustering etc). It is also recommended to use a MySQL Cluster in an HA environment to better handle the replication, failover, split-brain type scenarios

It is important to understand the basic differences so that you can make an informed choice about the best setup to use based on your specific environment. Depending on the configuration, you can replicate all/selected databases, or even selected tables within a database. In terms of the terminology:

- **Master**: is where all the changes happen. All database updates occur here, from adding, updating or deleting table records to creating functions, stored procedures or making table changes
- **Slave/Replica**: receives a copy of the changes applied at the master server. This all happens very quickly in order that the slave is always in sync with the master

Database Replication

There are three common types of replication, with pros/cons for each listed below:

- Master-Slave Replication
- Master-Master Replication
- MySQL Cluster

> **Please note**: The examples below are for illustrative purposes only. Please consult your Moogsoft technical contact for advice on the best strategy to use in your environment

Master-Slave Replication

One server acts as the master database and all other server(s) act as slaves. Writes can only occur on the master node by the application.

Pros	Cons
• Applications can read from the slave(s) without impacting the master • Backups of the entire database of relatively no impact on the master • Slaves can be taken offline and sync back to the master without any downtime	• In the instance of a failure a slave has to be promoted to master to take over its place. No automatic failover • Downtime and possibly lost of data when a master fails • All writes also have to be made to the master in a master-slave design • Each additional slave add some load* to the master since the binary log have to be read and data copied to each slave • Application might have to be restarted

Master-Master Replication

In a master-master configuration each master is configured as a slave to the other. Writes and reads can occur on both servers.

Pros	Cons
• Applications can read from both masters • Distributes write load across both master nodes • Simple, automatic and quick failover	• Loosely consistent • Not as simple as master-slave to configure and deploy

RabbitMQ (MooMS)

Refer to the standard RabbitMQ Clustering Guide documentation for the relevant details as needed. The following steps below show how to setup the clustering quickly between two servers. Assume that we are creating the cluster between two Cisco Crosswork Situation Manager servers, Moog 1 and Moog 2.

RabbitMQ Cluster Setup

Moog 1

- Check the status via:

```
service rabbitmq-server status
```

- If running, leave rabbitmq-server running on Server1

Moog 2

- Stop the rabbitmq-server via:

```
service rabbitmq-server stop
```

- Transfer the Erlang cookie located under /var/lib/rabbitmq from Server 1 to Server 2

- For two nodes to be able to communicate they must have the same shared secret called the Erlang cookie. The cookie is just a string of alphanumeric characters.Every cluster node must have the same cookie
- Erlang cookie output will be different, as it is generated automatically when Rabbit is started up for the first time

- Verify that the ownership of Erlang cookie is correct (owned by rabbitmq:rabbitmq)

```
# ls -al /var/lib/rabbitmq/
total 16
drwxr-xr-x   3 rabbitmq rabbitmq 4096 Jun 22 22:08 .
drwxr-xr-x. 20 root     root     4096 Jun 22 09:45 ..
-r--------   1 rabbitmq rabbitmq   20 Jun 22 22:21 .erlang.cookie
drwxr-x---   4 rabbitmq rabbitmq 4096 Jun 22 22:09 mnesia

# cat /var/lib/rabbitmq/.erlang.cookie
WRDEKOWQWLRQQDTXIIFT
```

- Compare the output of the cookie with the one on Server 1 to ensure that they match

To create the cluster, run the following commands:

Moog 2

```
# service rabbitmq-server start
Starting rabbitmq-server: SUCCESS
rabbitmq-server.

# rabbitmqctl stop_app
Stopping node rabbit@moog2 ...

# rabbitmqctl join_cluster  rabbit@moog1
Clustering node rabbit@moog2 with rabbit@moog1 ...

# rabbitmqctl start_app
Starting node rabbit@moog2 ...

# rabbitmqctl cluster_status
Cluster status of node rabbit@moog2 ...
[{nodes,[{disc,[rabbit@moog1,rabbit@moog2]}]},
 {running_nodes,[rabbit@moog1,rabbit@moog2]},
 {cluster_name,<<"rabbit@moog1.local">>},
 {partitions,[]}]
```

Moog 1

- Verify that it is setup properly:

```
[root@moog1 moogsoft]# rabbitmqctl cluster_status
Cluster status of node rabbit@moog1 ...
[{nodes,[{disc,[rabbit@moog1,rabbit@moog2]}]},
 {running_nodes,[rabbit@moog1]},
 {cluster_name,<<"rabbit@moog1.local">>},
 {partitions,[]}]
[root@moog1 moogsoft]#
```

Mirrored Queues

- Set queue mirroring policy to mirror all non-exclusive queues to all nodes. From either server (Moog 1 or Moog 2):

```
# rabbitmqctl set_policy -p <zone> ha-all ".+\.HA" '{"ha-mode":"all"}'
Setting policy "ha-all" for pattern ".+\\.HA" to "{\"ha-mode\":\"all\"}"
with priority "0" …
```

- On both Moog 1 and Moog 2, edit $MOOGSOFT-HOME/config/system.conf file and add both rabbitmq servers to the brokers object:

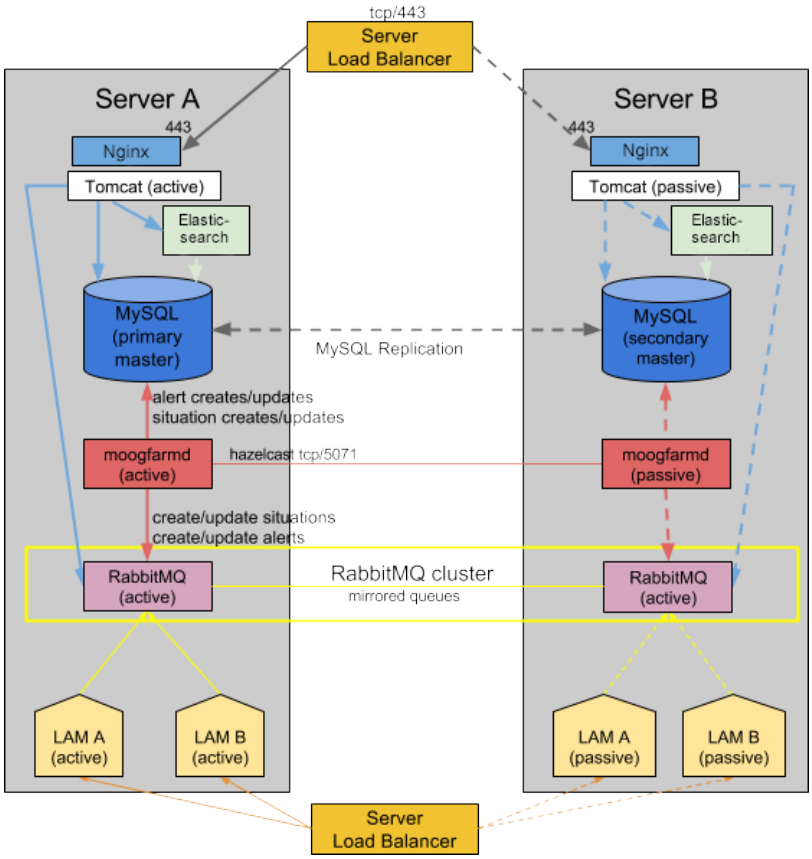
```
"brokers" :
[
    {
        "host"      : "Moog 1",
        "port"      : 5672
    },
    {
        "host"      : "Moog 2",
        "port"      : 5672
    }
]
```

Server Load Balancing

SLB configuration steps are not listed here. However, there are some example configuration steps listed based on HAProxy in the example deployment scenarios that can be referenced as needed. It is assumed that most enterprises have a dedicated team that handle SLB administration using their standard methods and procedures

For LAMs, configure healthcheck based on type of LAM, and whether any LAMs are expected to normally run Passive.

Two Server Active/Passive



Server A	Server B

• Tomcat (active)
• MySQL (primary master)
  • MySQL Replication slave to Server B
• moogfarmd (active)
  • Synced with moogfarmd on Server B via hazelcast
• RabbitMQ (active)
  • Clustered with RabbitMQ on Server B, with mirrored queues
• LAMs (active)
• Elasticsearch (active)
	• Tomcat (passive)
	• MySQL (secondary master)
	  • MySQL Replication slave to Server A (ready for failover)
	• moogfarmd (passive)
	  • Synced with moogfarmd on Server A via hazelcast
	• RabbitMQ (active)
	  • Clustered with RabbitMQ on Server A , with mirrored queues
	• LAMs (passive)
	• Elasticsearch (active) - will use MySQL on Server B (see replication config)

Sample Configuration

Cluster configuration template:

	Server A	Server B
Zone	zone_a	zone_a
Hostname	server_a	server_b
HA Cluster name	DC1	DC2
Cluster.Process Group.	moogfarmd: DC1.moog_farmd.moog_farmd-1	moogfarmd: DC2.moog_farmd.moog_farmd-1

Instance	Tomcat: DC1.UI.servlets Sample LAM: DC1.sample_lam.sample_lam-1	Tomcat: DC2.UI.servlets Sample LAM: DC2.sample_lam.rest_lam-1
system.conf	mooms.zone: zone_a mooms.brokers.host: server_a mooms.message_persistence: true mysql.host: server_a elasticsearch.host: server_a failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: DC1 & DC2 instances ha.cluster_name: DC1	mooms.zone: zone_a mooms.brokers.host: server_b mooms.message_persistence: true mysql.host: server_b elasticsearch.host: server_b failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: DC1 & DC2 instances ha.cluster_name: DC2

MySQL /etc/my.cnf (custom replication config)	`# Replication`
	`server-id=1`
	`log-bin=mysql-bin`
	`binlog_format=row`
	`replicate-ignore-table=moogdb.elastic_inc`
		`# Replication`
		`server-id=2`
		`log-bin=mysql-bin`
		`binlog_format=row`
		`replicate-ignore-table=moogdb.elasticsearch_inc`

RabbitMQ	`rabbitmqctl set_policy -p zone_a ha-all ".+\.HA" '{"ha-mode":"all"}'`	`rabbitmqctl stop_app`
		`rabbitmqctl join_cluster rabbit@server_a`
		`rabbitmqctl start_app`

Failover Scenarios

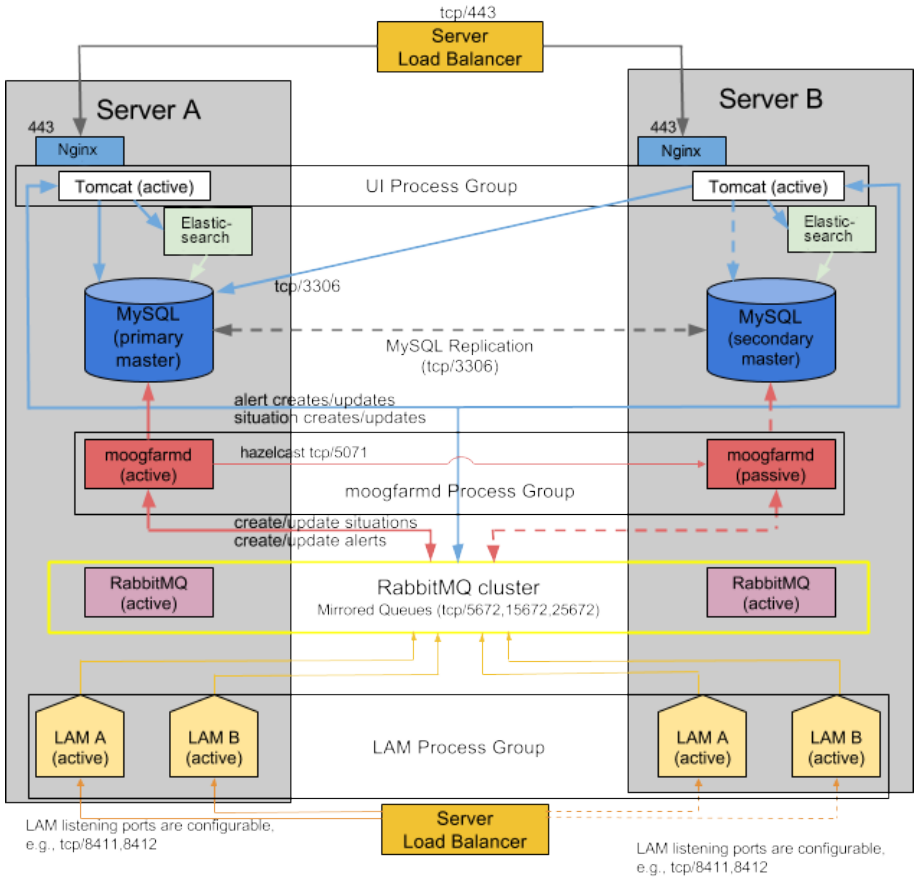
Server A Failure

- In the event Server A fails, ha_cntl is invoked to make moogfarmd, tomcat, and LAM components active on Server B. Components on Server B will use Server B's MySQL, RabbitMQ, and Elasticsearch. SLB will take Server A components (LAM, httpd, tomcat) out of the rotation, and put Server B components into the rotation
- Using ha_cntl to make Server B moogfarmd, tomcat, and LAM components active, based on the command: ha_cntl -a DC2

Post-failover Steps

- When restoring Server A, synchronize Server A's MySQL with Server B, and configure Server A as the Secondary in Master-Master setup
- Server A can be restored to Primary by using ha_cntl to switch processing back to Server A

Two Server Active/Active



Server A	Server B

• Tomcat (active)
• MySQL (primary master)
  • MySQL Replication slave to Server B
• moogfarmd (active)
  • Sync with moogfarmd on Server B via hazelcast
• RabbitMQ (active)
  • Clustered with RabbitMQ on Server B
• LAMs (active)
• Elasticsearch (active)
	• Tomcat (active)
	• MySQL (secondary master)
	  • MySQL Replication slave to Server A (ready for failover)
	• moogfarmd (passive)
	  • Sync with moogfarmd on Server A via hazelcast
	• RabbitMQ (active) clustered with RabbitMQ on Server A
	• LAMs (active)
	• Elasticsearch (active) - will use MySQL on Server B (see replication config)

Sample Configuration

Cluster configuration template:

	Server A	Server B
Zone	zone_a	zone_a

Hostname	server_a	server_b
HA Cluster name	DC	DC
Cluster.Process Group. **Instance**	moogfarmd: DC.moog_farmd.moog_farmd-1 Tomcat: DC.UI1.servlets (see note below) REST LAM: DC.rest_lam.rest_lam-1	moogfarmd: DC.moog_farmd.moog_farmd-2 Tomcat: DC.UI2.servlets (see note below) REST LAM: DC.rest_lam.rest_lam-2
system.conf	mooms.zone: zone_a mooms.brokers.host: server_a, server_b mooms.message_persistence: true mysql.host: server_a mysql.failover_connections: server_b elasticsearch.host: localhost failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: moog_farmd-1, servlets-1, rest_lam-1, etc ha.cluster_name: DC	mooms.zone: zone_a mooms.brokers.host: server_a, server_b mooms.message_persistence: true mysql.host: server_a mysql.failover_connections: server_b elasticsearch.host: localhost failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: moog_farmd-2, servlets-2, rest_lam-2, etc ha.cluster_name: DC

MySQL /etc/my.cnf (custom replication config)	`# Replication`
	`server-id=1`
	`log-bin=mysql-bin`
	`binlog_format=row`
	`replicate-ignore-table=moogdb.elasticsearch_inc`
		`# Replication`
		`server-id=2`
		`log-bin=mysql-bin`
		`binlog_format=row`
		`replicate-ignore-table=moogdb.elasticsearch_inc`

RabbitMQ	`rabbitmqctl set_policy -p zone_a ha-all ".+\.HA" '{"ha-mode":"all"}'`	`rabbitmqctl stop_app`
		`rabbitmqctl join_cluster rabbit@server_a`
		`rabbitmqctl start_app`

Note: As this is an active/active setup and message_persistence is set to true, the UI servlets must be run in separately named process groups.

Failover Scenarios

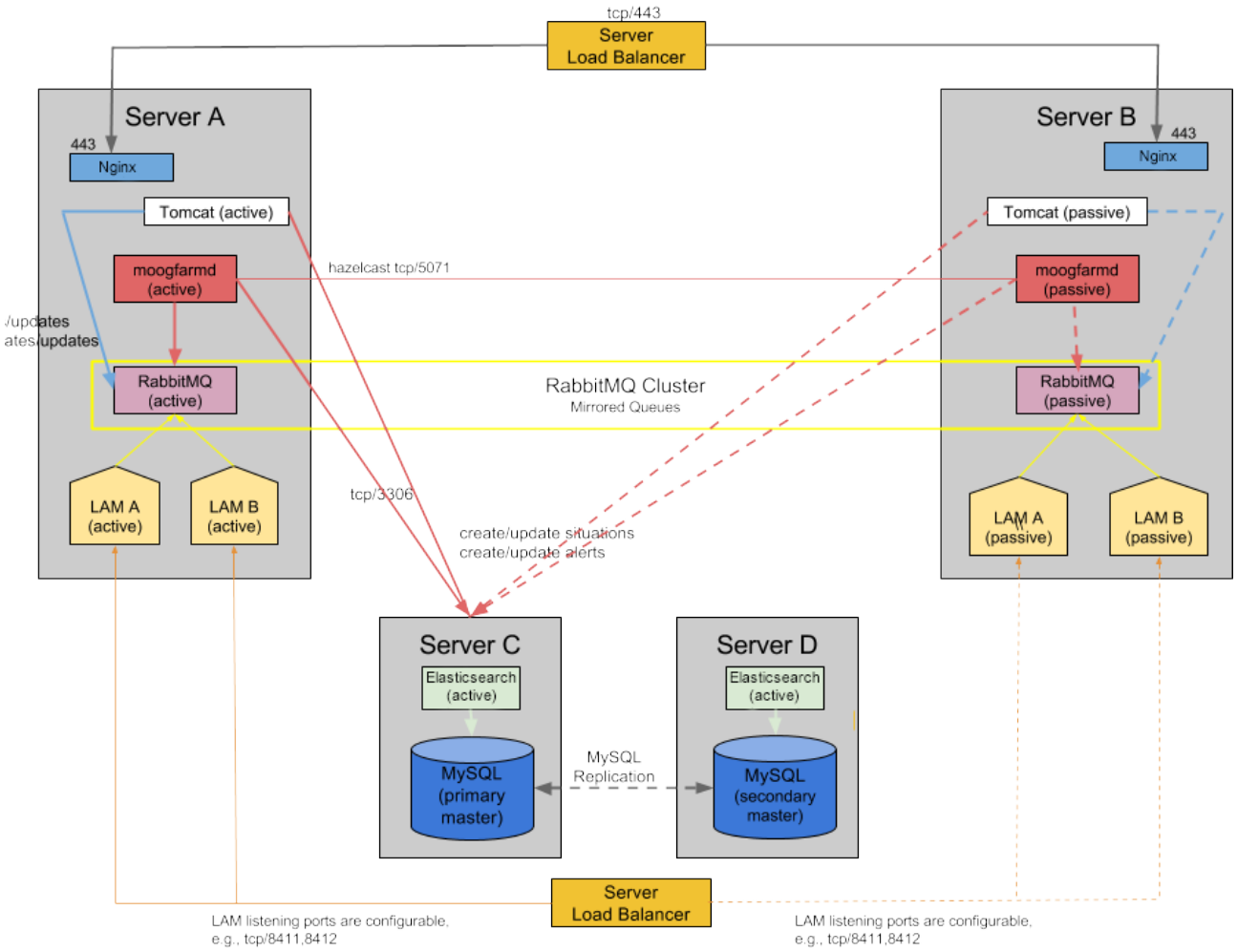
Server A Failure

- In the event Server A fails, ha_cntl can be invoked to make moogfarmd active on Server B. Components using MySQL will failover to Server B's MySQL, which will be considered Primary going forward. Components using Server A's RabbitMQ will switch over to Server B's RabbitMQ. SLB will take Server A components (LAM, httpd, tomcat) out of the rotation
- Using ha_cntl to make Server B moogfarmd active, based on the command: ha_cntl -a DC.moog_farmd.moog_farmd-2

Post-failover Steps

- Establish Server B MySQL as Primary: On each server, change system.conf's mysql.host to Server B, mysql.failover to Server A, and restart components (Moogfarmd can be kept running during these changes by switching which instance is active with ha_cntl, and only restart moogfarmd when it is passive!)
- When restoring Server A, synchronize Server A's MySQL with Server B, and configure Server A as the Secondary in Master-Master setup
- Server A can be restored to Primary by repeating the above steps, starting with shutting down Server B's MySQL, and letting components failover to Secondary on Server A (then change system.conf, restart components, etc.)

Four Server Active/Passive



Server A	Server B	Server C	Server D

• Tomcat (active)
• moogfarmd (active)
	• Tomcat (passive)
	• moogfarmd (passive)
		• MySQL (primary master)
		  • MySQL Replication slave to Server D
			• MySQL (secondary master)
			  • MySQL Replication slave to Server C (ready for failover)

• Synced with moogfarmd on Server B via hazelcast
• RabbitMQ (active)
  • Clustered with RabbitMQ on Server B, with mirrored queues
• LAMs (active)
	• Synced with moogfarmd on Server A via hazelcast
	• RabbitMQ (active)
	  • Clustered with RabbitMQ on Server A , with mirrored queues
	• LAMs (passive)
		• Elasticsearch(active)	• Elasticsearch (active)

Sample Configuration

Cluster configuration template:

	Server A	Server B
Zone	zone_a	zone_a
Hostname	server_a	server_b
HA Cluster name	DC1	DC2
Cluster.Process Group. **Instance**	moogfarmd: DC1.moog_farmd.moog_farmd-1 Tomcat: DC1.UI.servlets Sample LAM: DC1.sample_lam. sample_lam-1	moogfarmd: DC2.moog_farmd.moog_farmd-1 Tomcat: DC2.UI.servlets Sample LAM: DC2.sample_lam.rest_lam-1
system.conf	mooms.zone: zone_a mooms.brokers.host: server_a mooms.message_persistence: true mysql.host: server_c mysql.failover_connections: server_d elasticsearch.host: server_c failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: DC1 & DC2 instances ha.cluster_name: DC1	mooms.zone: zone_a mooms.brokers.host: server_b mooms.message_persistence: true mysql.host: server_c mysql.failover_connections: server_d elasticsearch.host: server_c failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: DC1 & DC2 instances ha.cluster_name: DC2

MySQL /etc/my.cnf (custom replication config)	`# Replication (Server C)`
	`server-id=1`
	`log-bin=mysql-bin`
	`binlog_format=row`
	`replicate-ignore-table=moogdb.elasticsearch_inc`
		`# Replication (Server D)`
		`server-id=2`
		`log-bin=mysql-bin`
		`binlog_format=row`
		`replicate-ignore-table=moogdb.elasticsearch_inc`

RabbitMQ	`rabbitmqctl set_policy -p zone_a ha-all`	`rabbitmqctl stop_app` `rabbitmqctl`

	`".+\.HA" '{"ha-mode":"all"}'`	`join_cluster rabbit@server_a` `rabbitmqctl start_app`

Failover Scenarios

Server A Failure

- In the event Server A fails, ha_cntl is invoked to make moogfarmd, tomcat, and LAM components active on Server B. Components on Server B will use Server B's RabbitMQ. SLB will take Server A components (LAM, httpd, tomcat) out of the rotation, and put Server B components into the rotation
- Using ha_cntl to make Server B moogfarmd, tomcat, and LAM components active, based on the command: ha_cntl -a DC2
- Server A can be restored to Primary by using ha_cntl to switch processing back to Server A

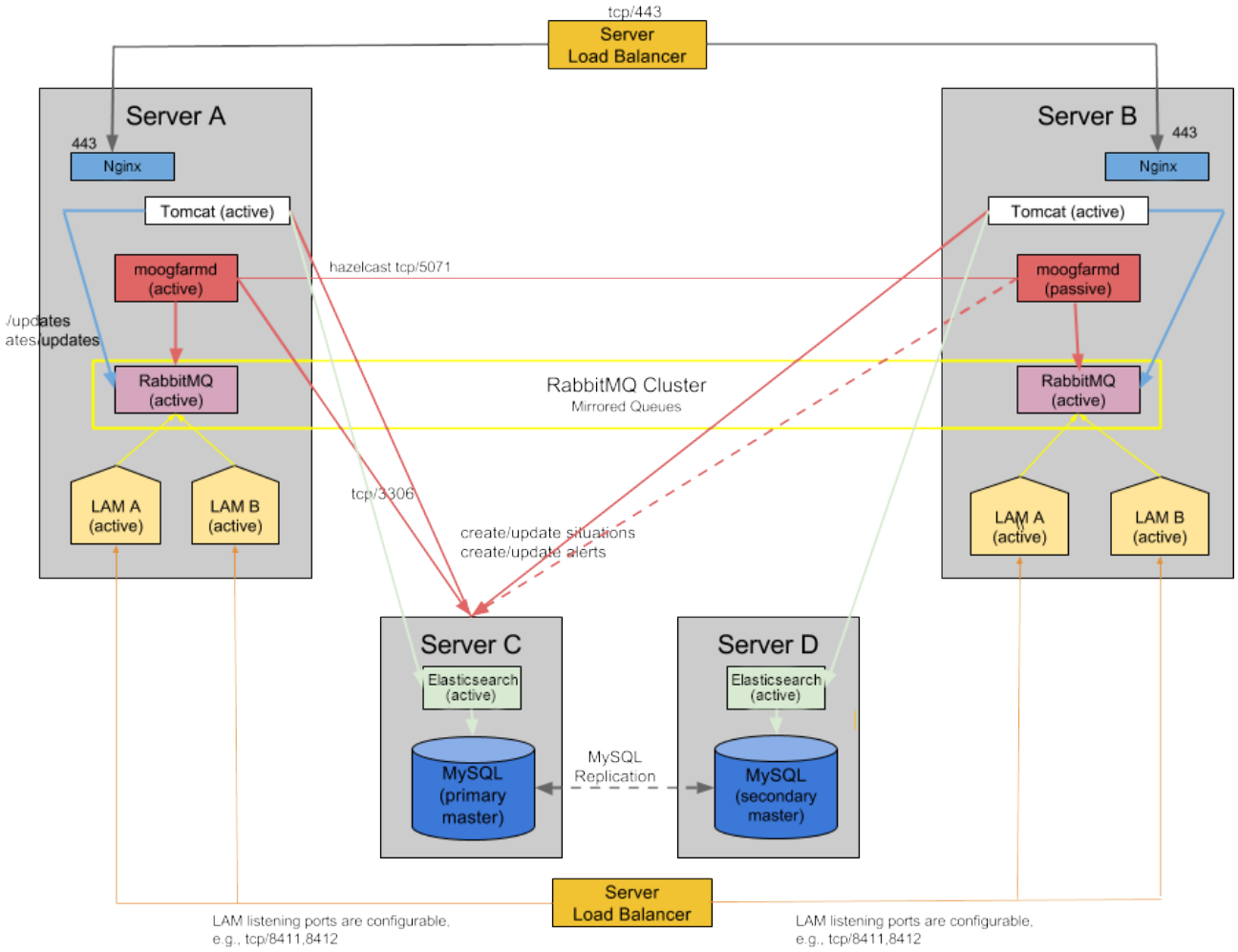
Server C Failure

- In the event Server C fails, active components will use their mysql.failover_connection setting to connect to Server D
- elasticsearch failover need to be done manually by changing system.conf

Post-failover Steps

- Before restoring Server C, change system.conf so that Server D is primary mysql connection, Server C is failover
- When restoring Server C, synchronize Server C's MySQL with Server D, and configure Server C as the Secondary in Master-Master setup
- Server C can be restored to primary master by shutting down Server D, allowing components to failover to Server C, then reverse primary /failover in system.conf again. Last, bring up mysql on Server D, resync as slave to Server C

Four Server Active/Active



Server A	Server B	Server C	Server D

• Tomcat (active)
• moogfarmd (active)
  • Synced with moogfarmd on Server B via hazelcast
• RabbitMQ (active)
  • Clustered with RabbitMQ on Server B, with mirrored queues
• LAMs (active)
	• Tomcat (active)
	• moogfarmd (passive)
	  • Synced with moogfarmd on Server A via hazelcast
	• RabbitMQ (active)
	  • Clustered with RabbitMQ on Server A , with mirrored queues
	• LAMs (active)
		• MySQL (primary master)
		  • MySQL Replication slave to Server D
		• Elasticsearch (active)
			• MySQL (secondary master)
			  • MySQL Replication slave to Server C (ready for failover)
			• Elasticsearch (active)

Sample Configuration

Cluster configuration template:

	Server A	Server B
Zone	zone_a	zone_a
Hostname	server_a	server_b
HA Cluster name	DC1	DC2

Cluster.Process Group. **Instance**	moogfarmd: DC1.moog_farmd.moog_farmd-1 Tomcat: DC1.UI1.servlets (see note below) Sample LAM: DC1.sample_lam.sample_lam-1	moogfarmd: DC2.moog_farmd.moog_farmd-1 Tomcat: DC2.UI2.servlets (see note below) Sample LAM: DC2.sample_lam.rest_lam-1
system.conf	mooms.zone: zone_a mooms.brokers.host: server_a, server_b mooms.message_persistence: true mysql.host: server_c mysql.failover_connections: server_d elasticsearch.host: server_c failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: DC1 & DC2 instances ha.cluster_name: DC1	mooms.zone: zone_a mooms.brokers.host: server_a, server_b mooms.message_persistence: true mysql.host: server_c mysql.failover_connections: server_d elasticsearch.host: server_d failover.persist_state: true failover.hazelcast.hosts: server_a, server_b process_monitor: DC1 & DC2 instances ha.cluster_name: DC2

MySQL /etc/my.cnf (custom replication config)	`# Replication (Server C)`
	`server-id=1`
	`log-bin=mysql-bin`
	`binlog_format=row`
	`replicate-ignore-table=moogdb.elasticsearch_inc`
		`# Replication (Server D)`
		`server-id=2`
		`log-bin=mysql-bin`
		`binlog_format=row`
		`replicate-ignore-table=moogdb.elasticsearch_inc`

RabbitMQ	`rabbitmqctl set_policy -p zone_a ha-all ".+\.HA" '{"ha-mode":"all"}'`	`rabbitmqctl stop_app`
		`rabbitmqctl join_cluster rabbit@server a`
		`rabbitmqctl start_app`

Note: As this is an active/active setup and message_persistence is set to true, the UI servlets must be run in separately named process groups.

Failover Scenarios

Server A Failure

- In the event Server A fails, ha_cntl can be invoked to make moogfarmd active on Server B. Components using Server A's RabbitMQ will switch over to Server B's RabbitMQ. SLB will take Server A components (LAM, httpd, tomcat) out of the rotation
- Using ha_cntl to make Server B moogfarmd active, based on the command: ha_cntl -a DC.moog_farmd.moog_farmd-2

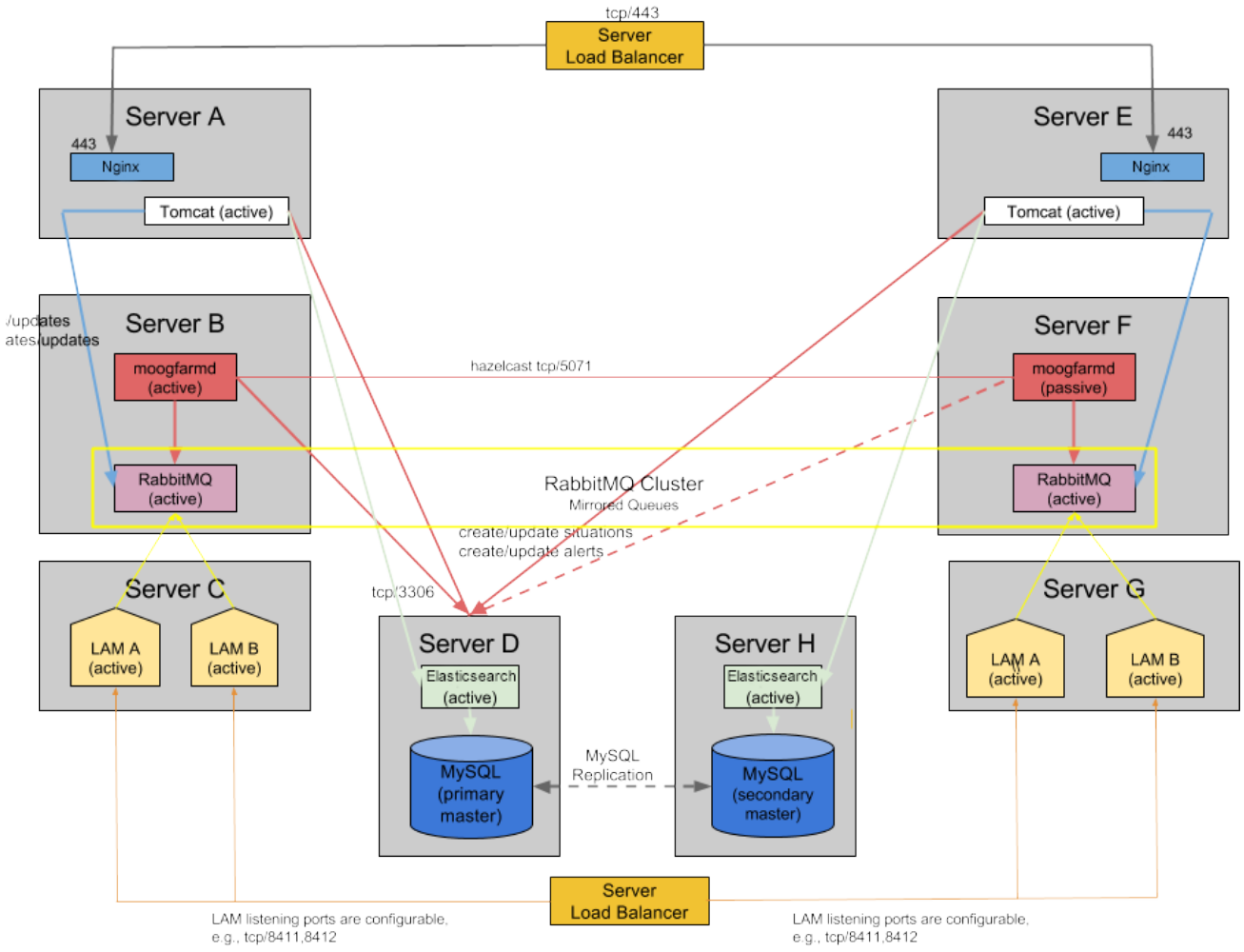
Server C Failure

- In the event Server C fails, active components will use their mysql.failover_connection setting to connect to Server D
- elasticsearch failover needs to be done manually by changing system.conf

Post-failover Steps

- Establish Server D MySQL as Primary: On each server, change system.conf's mysql.host to Server D, mysql.failover to Server C, and restart components (Moogfarmd can be kept running during these changes by switching which instance is active with ha_cntl, and only restart moogfarmd when it is passive!)
- When restoring Server C, synchronize Server C's MySQL with Server D, and configure Server C as the Secondary in Master-Master setup
- Server C can be restored to Primary by repeating the above steps, starting with shutting down Server D's MySQL, and letting components failover to Secondary on Server C (then change system.conf, restart components, etc)
- Change Server A's system.conf so that Server A's tomcat fails over to use Server D's elasticsearch. Server B's tomcat already uses Server D's elasticsearch, so no change, no failover. Server D elasticsearch's connection to Server D's DB does not change either

Eight Server Active/Active Fully Distributed

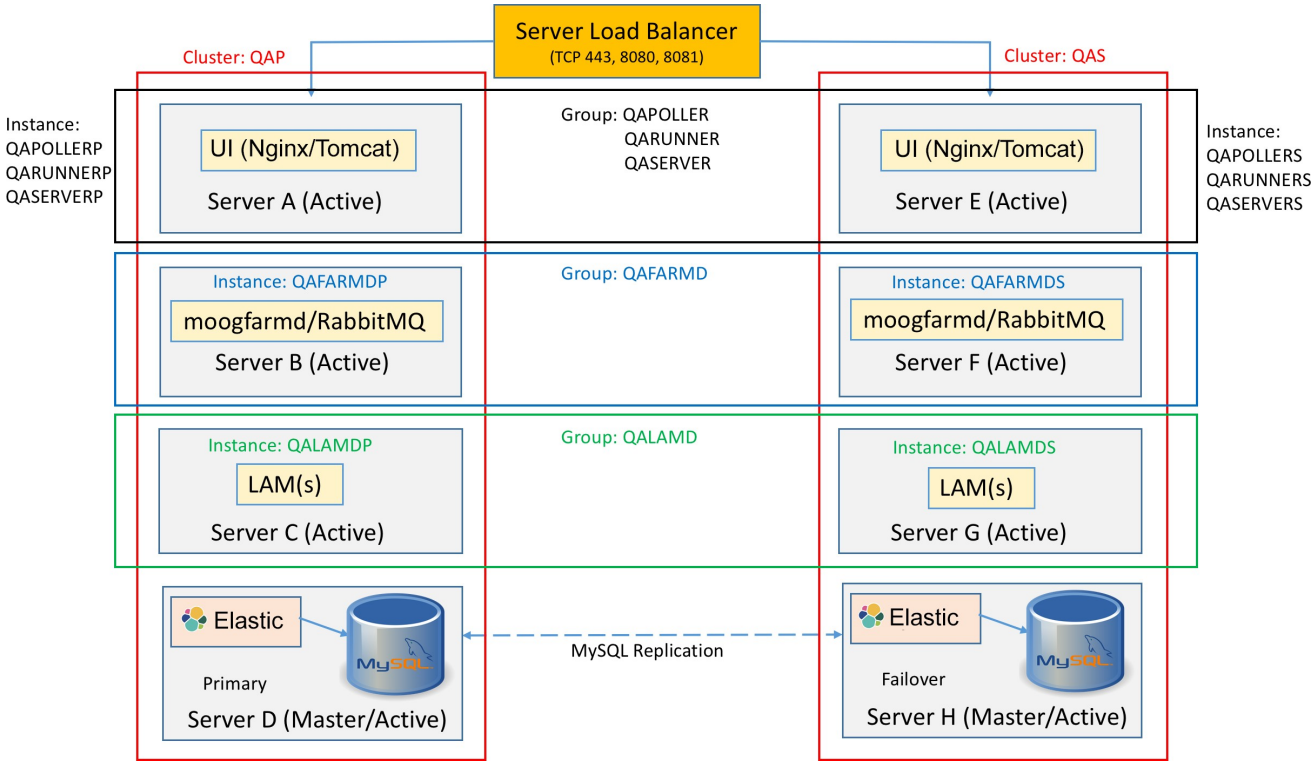


Server A, Server E	Server B, Server F	Server C, Server G	Server D, Server H
• Moog packages installed: moogsoft-ui		• Moog packages installed: moogsoft-lams	• Moog packages installed: moogsoft-db, moogsoft-search

• Tomcat (active) and nginx on both servers	• Moog packages installed: moogsoft-mooms + moogsoft-server + moogsoft-utils
	• moogfarmd (active) on Server B
	  • Synced with moogfarmd (Passive) on Server F via Hazelcast
	• RabbitMQ (active)
	  • Clustered with RabbitMQ on Server F, with mirrored queues
		• LAMs (Active) on both Servers. Can be used with/behind a Load Balancer or without	• Elasticsearch (Active) on both Servers. Note that Elasticsearch currently do not support HA
			• MySQL (Server D) is the Primary Master/Active
			• MySQL (Server H) is the Failover Master/Active
			• Both MySQL Servers are setup for bi-directional Active/Active replication

HA Cluster, Group, Instance Visualization

The diagram shows how the HA Cluster, Group and Instances are setup across all eight servers.



Sample Configuration

- The sample configuration below only show the relevant portion of the moog config files needed for the HA setup as shown in the above architecture diagram. It does not list the complete config output of each file
- The sample configuration below also does not show the Server Load Balancer (SLB) Configuration either. Depending upon the SLB used (for example HAProxy, F5 etc.), refer to the respective documentation
- Moog documentation also have some example HAProxy configuration listed that can be used as reference/applicable

UI Servers

$MOOGSOFT_HOME/config/servlets.conf	
Server A	**Server E**

```
{
   loglevel: "WARN",
   webhost :
"https://<Server_Load_Balance
r_VIP_or_Name>",
   moogsvr:
   {
        eula_per_user: false,
        cache_root: "/mnt/nfs
/shared",

db_connections:        10,

priority_db_connections: 25
    },
    moogpoller :
    {
    },
    toolrunner :
    {
        sshtimeout: 900000,
        toolrunnerhost:
"Server D",
        toolrunneruser:
"<toolrunner username>",
        toolrunnerpassword:
"<toolrunner password>"
    },
    graze  :
    {
    },
    events  :
    {
    },
    ha :
    {
        cluster: "QAP",
        instance: "SERVLETS-
P",
        group: "UI",
        start_as_passive:
false
    }
}
```

```
{
   loglevel: "WARN",
   webhost :
"https://<Server_Load_Balance
r_VIP_or_Name>",
   moogsvr:
   {
        eula_per_user: false,
        cache_root: "/mnt/nfs
/shared",

db_connections:        10,

priority_db_connections: 25
    },
    moogpoller :
    {
    },
    toolrunner :
    {
        sshtimeout: 900000,
        toolrunnerhost:
"Server H",
        toolrunneruser:
"<toolrunner username>",
        toolrunnerpassword:
"<toolrunner password>"
    },
    graze  :
    {
    },
    events  :
    {
    },
    ha :
    {
        cluster: "QAS",
        instance: "SERVLETS-
S",
        group: "UI",
        start_as_passive:
false
    }
}
```

$MOOGSOFT_HOME/config/system.conf

Server A

```
"mooms" :
        {
```

Server E

```
"mooms" :
        {
```

```
            #
            "zone" : "QA",
            "brokers" : [
                {
                    #
                    # This
can be an IPv4 or IPv6
address
                    #
                    "host" :
"Server B",
                     #
                    # The
listening port of the MooMS
broker
                    #
                    "port":
5672
                }
            ],
 "mysql" :
            {
                "host" :
"Server D",

"database"            :
"moogdb",
                "port" :
3306,

                # To use
Multi-Host Connections for
failover support use:
                #

"failover_connections":
                   [
                     {

"host": "Server H",

"port": 3306
                     }
"search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server D",

"port"     : 9200
```

```
            #
            "zone" : "QA",
             "brokers" : [
                {
                    #
                    # This
can be an IPv4 or IPv6
address
                    #
                    "host" :
"Server F",
                     #
                    # The
listening port of the MooMS
broker
                    #
                    "port":
5672
                }
            ],
"mysql" :
            {
                "host" :
"Server D",

"database"            :
"moogdb",
                "port" :
3306,
                # To use
Multi-Host Connections for
failover support use:
                #

"failover_connections":
                   [
                    {

"host": "Server H",

"port": 3306
                     }
"search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server H",

"port"     : 9200
                }
```

```
                }
            ]
        },

# Failover configuration
parameters.
        "failover":
            {
                # Set
persist_state to true to
turn persistence on. If set,
state
                # will be
persisted in a Hazelcast
cluster.

"persist_state" : true,
        "process_monitor":
# Moog_farmd
                  {

"group"           :
"moog_farmd",

"instance"        :
"QAFARMDP",
                  # Servlets
                     {

"group"           : "UI",

"instance"        :
"SERVLETS-P",

"service_name"    : "apache-
tomcat",

"process_type"    :
"servlets",

"reserved"        : true,

"subcomponents"   : [

"moogsvr",

"moogpoller",

"toolrunner"

]
                  },
                   # Lams
```

```
            ]
        },
# Failover configuration
parameters.
        "failover":
            {
                # Set
persist_state to true to
turn persistence on. If set,
state
                # will be
persisted in a Hazelcast
cluster.

"persist_state" : true,
        "process_monitor":

# Moog_farmd
                  {

"group"           :
"moog_farmd",

"instance"        :
"QAFARMDS",
                  # Servlets
                  {

"group"           : "UI",

"instance"        :
"SERVLETS-S",

"service_name"    : "apache-
tomcat",

"process_type"    :
"servlets",

"reserved"        : true,

"subcomponents"   : [

"moogsvr",

"moogpoller",

"toolrunner"

]
                     },
                   # Lams
                   {
```

```
                        {

"group"                 :
"rest_lam",

"instance"          :
"QALAMDP",

"service_name"      :
"rest_lamd",

"process_type"      : "LAM",

"reserved"              : false
            "ha":
            {
                # Use this
to change the default HA
cluster name.
                "cluster":
"QAP"
            }
```

```
"group"             :
"rest_lam",

"instance"          :
"QALAMDS",

"service_name"      :
"rest_lamd",

"process_type"      : "LAM",

"reserved"          : false
            "ha":
            {
                # Use this
to change the default HA
cluster name.
                "cluster":
"QAS"
            }
```

Moogfarmd/RabbitMQ Servers

$MOOGSOFT_HOME/config/system.conf

Server B

```
"mooms" :
    {
        "zone" : "QA",
        "brokers" : [
            {
                #
                # This can
be an IPv4 or IPv6 address
                #
                "host" :
"Server B",
                #
                # The
listening port of the MooMS
broker
                #
                "port" : 5672
            },
            {
                #
                # This can
be an IPv4 or IPv6 address
```

Server F

```
    "mooms" :
    {
        "zone" : "QA",
        "brokers" : [
            {
                #
                # This can
be an IPv4 or IPv6 address
                #
                "host" :
"Server B",

                #
                # The
listening port of the MooMS
broker
                #
                "port" : 5672
            },
            {
                #
                # This can
```

```
                #
                "host" :
"Server C",
                #
                # The
listening port of the MooMS
broker
                #
                "port" : 5672
            }
        ],

"message_persistence" : true,
    "mysql" :
        {
            "host" : "Server
D",


"database"        : "moogdb",

"failover_connections" :
               [
                 {
                     "host"
: "Server H",
                    "port"
: 3306
                 }
    "search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server D",

"port"     : 9200
                }
            ]
        },
    # Failover configuration
parameters.
    "failover" :
        {
            # Set
persist_state to true to
turn persistence on. If set,
state
            # will be
persisted in a Hazelcast
cluster.
```

```
be an IPv4 or IPv6 address
                #
                "host" :
"Server C",

                #
                # The
listening port of the MooMS
broker
                #
                "port" : 5672
            }
        ],

"message_persistence" : true,
    "mysql" :
        {
            "host" : "Server
D",

"database"        : "moogdb",

"failover_connections" :
               [
                 {
                     "host"
: "Server H",
                     "port"
: 3306
                 }
    "search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server H",

"port"     : 9200
                }
            ]
        },

    # Failover configuration
parameters.
    "failover" :
        {
            # Set
persist_state to true to
turn persistence on. If set,
state
            # will be
```

```
            "persist_state"
: true,
            # Configuration
for the Hazelcast cluster.
            "hazelcast" :
                {
                    # A list
of hosts to allow to
participate in the cluster.
                    "hosts"
: ["Server B", "Server F"],
                    #
Moog_farmd
                                {

"group"           :
"moog_farmd",

"instance"        :
"QAFARMDP",

"service_name"    :
"moogfarmd",

"process_type"    :
"moog_farmd",

"reserved"        : true,

"subcomponents"   : [

"AlertBuilder",

"Sigaliser",

"Default Cookbook",

"Journaller",

"TeamsMgr"

#"AlertRulesEngine",

#"SituationMgr",

#"Notifier"

]

},

# Servlets
                                {
```

```
persisted in a Hazelcast
cluster.
            "persist_state"
: true,

            # Configuration
for the Hazelcast cluster.
            "hazelcast" :
                {
                    # A list
of hosts to allow to
participate in the cluster.
                    "hosts"
: ["Server B", "Server F"],
                    #
Moog_farmd
                                {

"group"           :
"moog_farmd",

"instance"        :
"QAFARMDS",

"service_name"    :
"moogfarmd",

"process_type"    :
"moog_farmd",

"reserved"        : true,

"subcomponents"   : [

"AlertBuilder",

"Sigaliser",

"Default Cookbook",

"Journaller",

"TeamsMgr"

#"AlertRulesEngine",

#"SituationMgr",

#"Notifier"

]

},
```

```
"group"           : "UI",

"instance"        :
"SERVLETS-P",

"service_name"    : "apache-
tomcat",

"process_type"    :
"servlets",

"reserved"        : true,

"subcomponents"   : [

"moogsvr",

"moogpoller",

"toolrunner"

]
                      },
                      },

# Lams
                               {

"group"           :
"rest lam",

"instance"        :
"QALAMDP",

"service_name"    :
"rest lamd",

"process type"    : "LAM",

"reserved"        : false
                               }
                          ]
        }
        "ha":
        {
            # Use this to
change the default HA
cluster name.
            "cluster": "QAP"
        }
```

```
# Servlets
                               {

"group"           : "UI",

"instance"        :
"SERVLETS-S",

"service_name"    : "apache-
tomcat",

"process_type"    :
"servlets",

"reserved"        : true,

"subcomponents"   : [

"moogsvr",

"moogpoller",

"toolrunner"

]
                          },

# Lams
                               {

"group"           :
"rest_lam",

"instance"        :
"QALAMDS",

"service_name"    :
"rest_lamd",

"process_type"    : "LAM",

"reserved"        : false
                               }
                          ]
        }
        "ha":
        {
            # Use this to
change the default HA
cluster name.
            "cluster": "QAS"
```

```
        }
```

$MOOGSOFT_HOME/config/moog_farmd.conf	
Server B	**Server F**
For all the Moolets that are being used and are set to "run_on_startup: true", ensure that they have "persist_state: true" to take advantage of message persistence in case of a failover.	For all the Moolets that are being used and are set to "run_on_startup: true", ensure that they have "persist_state: true" to take advantage of message persistence in case of a failover.

`ha:`
`{`
`cluster: "QAP",`
`group: "QAFARMD",`
`default_leader: true`
`}`
	`ha:`
	`{`
	`cluster: "QAS",`
	`group: "QAFARMD",`
	`default_leader: true`
	`}`

LAM Servers

$MOOGSOFT_HOME/config/system.conf	
Server C	**Server G**

`"mooms" :`
`{`
`"zone" : "QA",`
 	
`"brokers" : [`
`{`
`#`
`# This can be an IPv4 or IPv6 address`
`#`
`"host" : "Server B",`
 	
`#`
`# The listening port of the MooMS broker`
`#`
`"port" : 5672`
 	
`"message_persistence" : true,`
`"mysql" :`
`{`
`"host" : "Server D",`
	`"mooms" :`
	`{`
	`"zone" : "QA",`
	
	`"brokers" : [`
	`{`
	`#`
	`# This can be an IPv4 or IPv6 address`
	`#`
	`"host" : "Server F",`
	
	`#`
	`# The listening port of the MooMS broker`
	`#`
	`"port" : 5672`
	`"mysql" :`
	`{`
	`"host" : "Server D",`
	
	`"database" : "moogdb",`

```
"database"        : "moogdb",

"username"        :
"moogsoft",

"password"        : "m00",
#

"failover_connections" :
               [
                 {
                     "host"
: "Server H",
                     "port"
: 3306
                 }
"search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server D",

"port"     : 9200
                }
            ]
        },
      # Persistence
configuration parameters.
    "persistence" :
        {
            # Set
persist_state to true to
turn persistence on. If set,
state
            # will be
persisted in a Hazelcast
cluster.
            "persist_state"
: true,
# Moog_farmd
                              {

"group"           :
"moog_farmd",

"instance"        :
"QAFARMDP",

"service_name"    :
```

```
"username"        :
"moogsoft",

"password"        : "m00",

"failover_connections" :
               [
                 {
                     "host"
: "Server H",
                     "port"
: 3306
                 }
"search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server H",

"port"     : 9200
                }
            ]
        },
# Persistence configuration
parameters.
    "persistence" :
        {
            # Set
persist_state to true to
turn persistence on. If set,
state
            # will be
persisted in a Hazelcast
cluster.
            "persist_state"
: true,
# Moog_farmd
                              {

"group"           :
"moog_farmd",

"instance"        :
"QAFARMDS",

"service_name"    :
"moogfarmd",

"process_type"    :
```

```
"moogfarmd",
"process_type"    :
"moog_farmd",
"reserved"        : true,
"subcomponents"   : [
"AlertBuilder",
"Default Cookbook",
"Journaller",
"TeamsMgr"
#"AlertRulesEngine",
#"SituationMgr",
#"Notifier"
]
},

# Servlets                          {
"group"           : "UI",
"instance"        :
"SERVLETS-P",
"service_name"    : "apache-
tomcat",
"process_type"    :
"servlets",
"reserved"        : true,
"subcomponents"   : [
"moogsvr",
"moogpoller",
"toolrunner"
]
```

```
"moog_farmd",
"reserved"        : true,
"subcomponents"   : [
"AlertBuilder",
"Sigaliser",
"Default Cookbook",
"Journaller",
"TeamsMgr"
#"AlertRulesEngine",
#"SituationMgr",
#"Notifier"
]
},

# Servlets                          {
"group"           : "UI",
"instance"        :
"SERVLETS-S",
"service_name"    : "apache-
tomcat",
"process_type"    :
"servlets",
"reserved"        : true,
"subcomponents"   : [
"moogsvr",
"moogpoller",
"toolrunner"
]
                        },
```

```
                        },

# Lams
                                {

"group"             :
"rest_lam",

"instance"          :
"QALAMDP",

"service_name"      :
"rest_lamd",

"process_type"      : "LAM",

"reserved"          : false
                                }
"ha":
        {
            # Use this to
change the default HA
cluster name.
            "cluster": "QAP"
        }
```

```
# Lams
                                {

"group"             :
"rest_lam",

"instance"          :
"QALAMDS",

"service_name"      :
"rest_lamd",

"process_type"      : "LAM",

"reserved"          : false
                                }
"ha":
        {
            # Use this to
change the default HA
cluster name.
            "cluster": "QAS"
        }
```

$MOOGSOFT_HOME/config/rest_lam.conf

Server C

```
ha:
     {
        cluster: "QAP",
        group: "QALAMD"
     },
```

Server G

```
 ha:
     {
        cluster: "QAS",
        group: "QALAMD"
     },
```

Database Servers

$MOOGSOFT_HOME/config/system.conf

Server D

```
"mysql" :
        {

"host"              :
"localhost",

"database"          : "moogdb",
```

Server H

```
"mysql" :
        {

"host"              :
"localhost",

"database"          : "moogdb",
```

```
"port"            : 3306
        },

"search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server D",

"port"     : 9200
                }
            ]
        },
 "ha":
        {
            # Use this to
change the default HA
cluster name.
            "cluster": "QAP"
        }
```

```
"port"            : 3306
       },

"search" :
        {
            "limit"
: 1000,
            "nodes" : [
                {

"host"     : "Server H",

"port"     : 9200
                }
            ]
        },
 "ha":
        {
            # Use this to
change the default HA
cluster name.
            "cluster": "QAS"
        }
```

NFS Shared Storage setup for handling server files

File attachments uploaded to the UI (Situation Room thread entry attachments and User Profile Pictures) are written to the disk on the UI server to which the user is connected to - therefore to ensure these attachments are available to users logged in to other UI servers as part of an HA /Load Balancer setup, the location of these attachments should be configured to be on a shared disk (NFS) available to all UI servers. The example below show a sample configuration.

- The configuration below assume that /mnt/nfs/shared is the shared location exposed by the NFS Server called NFS_Server
- Ensure that the tomcat:tomcat user/group has the same uid/gid across all 3 servers and has ownership of the shared directory. If not then the following commands can be run on all 3 servers to ensure that they are the same. The gid can be be verified on all 3 servers via: cat /etc/group | grep tomcat command

```
groupmod -g <gid> tomcat
usermod -u <gid> tomcat
```

NFS Server

```
service apache-tomcat stop

chown -R tomcat:tomcat /usr/share/apache-tomcat

chown -R tomcat:tomcat /var/run/apache-tomcat

mkdir /shared

chown tomcat:tomcat /shared/
```

```
chmod 755 /shared/

yum install -y nfs-utils nfs-utils-lib
chkconfig nfs on

service rpcbind start

service nfs start

service apache-tomcat start

Edit /etc/exports and set: /shared Server A(rw,sync,no_subtree_check,
insecure) Server E(rw,sync,no subtree check,insecure)

Then run: exportfs -a
```

Server A

```
service apache-tomcat stop

chown -R tomcat:tomcat /usr/share/apache-tomcat

chown -R tomcat:tomcat /var/run/apache-tomcat

yum install -y nfs-utils nfs-utils-lib

mkdir -p /mnt/nfs/shared

mount NFS_Server:/shared /mnt/nfs/shared

Edit $MOOGSOFT_HOME/config/servlets.conf and set cache_root: "/mnt/nfs
/shared",

service apache-tomcat start
```

Server E

```
service apache-tomcat stop

chown -R tomcat:tomcat /usr/share/apache-tomcat

chown -R tomcat:tomcat /var/run/apache-tomcat

yum install -y nfs-utils nfs-utils-lib

mkdir -p /mnt/nfs/shared

mount NFS Server:/shared /mnt/nfs/shared
```

```
Edit $MOOGSOFT_HOME/config/servlets.conf and set cache_root: "/mnt/nfs
/shared",

service apache-tomcat start
```

Failover Scenarios

- moog_farmd: moog_farmd process running in Passive mode will not process Events or detect Situations. When it fails over to Active mode, it will be able to carry on using the state from the previously Active Instance if this has been persisted
- LAMs operating in Passive mode do not send Events to the MooMS bus. For example, the REST LAM in Passive mode will reject POST requests with an HTTP status of 503 (server unavailable)

Moog_farmd and UI Failover

- In the setup shown in Figure A, all components are running in active/active mode except for moog_farmd
- In the event Server B fails, ha_cntl can be invoked to make moogfarmd active on Server F. Components using Server B's RabbitMQ will switch over to Server F's RabbitMQ. Since both UI's are active behind a SLD, users will see no delay as there is no switch-over from one UI to another in this scenario. They will seamlessly pick-up from Server F
- Using ha_cntl to make Server F moogfarmd active, based on the command: ha_cntl -a QAP.QAFARMD
- Since both moog_farmd's are in the same HA group (Figure B), activating the primary on Server B, will automatically de-activate the moog_farmd on Server F (making it passive). To switch over to Server F, ha_cntl -a QAS.QAFARMD will activate moog_farmd on Server F making the Server B's inactive
- Use the "ha_cntl -v" output to see the HA status of the cluster

LAMs

- Since both LAMs are active and behind a SLB, only one LAM will be sending events onto the message bus, thus preventing sending duplicate events. In case of Server C LAM failure, SLB will seamlessly transition the data flow over to Server G

- It is important to note that in case where both LAMs on Server C and G are not behind a SLB and events are sent to both at the same time, will result in sending duplicate events onto the message bus, which will cause moog_farmd seeing 2 events and will create 2 Alerts (2 LAMs = 2 Events on the bus = 2 Alerts). This will result in users seeing duplicate Alerts in the UI, which will have the same details (date/time stamp, description) with different Alert IDs. To prevent this, only one LAM can be running in active mode and the other must be in passive mode
- Since both LAMs are in the same HA group (Figure B), starting the LAM on Server C via ha_cntl -a QAP.QALAMD will automatically de-activate the LAM on Server G making it passive. To switch over to the LAM on Server G, ha_cntl -a QAS.QALAMD will make it active, thereby making the other on Server C as passive

Database and Elasticsearch

- Server D is the active DB with Server H as the failover connection with active/active replication between them. Both elasticsearch are setup to point to their respective DB Servers. In case of Server D failing, the DB connection will automatically failover to Server H. Whilst the failover is occurring some temporary MySQL connection errors or warnings may be seen in the log output. If should be noted that if the primary or another failover_connection higher up the list becomes available again, the connection will not automatically failback to that until the a moog component is restarted or makes a new connection
- Due to the possibility of DB split-brain scenario (i.e. the two DBs can no longer talk to each other, but they are still running), it is recommended to use a MySQL Cluster instead

MySQL Bi-Directional Replication Configuration

Relevant Commands

- show processlist
- show slave status
- show master status

Server D	Server H
• Setup my.cnf file	• Setup my.cnf file

```
/etc/my.cnf:
# Replication
server-id=1
log-bin=mysql-bin
binlog_format=row
replicate-ignore-
table=moogdb.
elasticsearch_inc
```

- Create Replication user

```
mysql> CREATE USER
'repl'@'Server H' IDENTIFIED
BY 'm00g';
mysql> GRANT REPLICATION
SLAVE ON *.* TO
'repl'@'Server H';
```

- Lock Tables and Record file/position

```
mysql> FLUSH TABLES WITH
READ LOCK;
mysql -u root -e "SHOW
MASTER STATUS;"  (in
separate session, output
includes file & position)
```

- Create Copy of Master DB

```
mysqldump --all-databases --
master-data > dbdump.db
```

- Quit mysql session above that had read lock after starting mysqldump

```
/etc/my.cnf:
# Replication
server-id=2
log-bin=mysql-bin
binlog_format=row
replicate-ignore-
table=moogdb.
elasticsearch_inc
```

- Create Replication user

```
mysql> CREATE USER
'repl'@'Server D' IDENTIFIED
BY 'm00g';
mysql> GRANT REPLICATION
SLAVE ON *.* TO
'repl'@'Server D';
```

- Load Copy of Master DB

```
mysql < dbdump.db
```

- Configure and Start Slave

```
mysql> CHANGE MASTER TO
MASTER_HOST='Server D',
MASTER_USER='repl',
MASTER_PASSWORD='m00g',
MASTER_LOG_FILE='<file>',
MASTER_LOG_POS=<position>;
```

Example:

```
mysql> CHANGE MASTER TO
MASTER_HOST='Server D',
MASTER_USER='repl',
MASTER_PASSWORD='m00g',
MASTER_LOG_FILE='mysql-bin.
000001', MASTER_LOG_POS=6144;
mysql> START SLAVE;
mysql> SHOW SLAVE STATUS\G;
```

Complete Master/Master Setup

Server H	Server D (using Server H's file & position)

```
mysql> show master status;
```

```
mysql> CHANGE MASTER TO
MASTER_HOST=Server H,
MASTER_USER='repl',
MASTER_PASSWORD='m00g',
MASTER_LOG_FILE='mysql-bin.
000003',
MASTER_LOG_POS=27639080;

mysql> START SLAVE;

mysql> SHOW SLAVE STATUS\G;
```

You may have to use: “mysqladmin -u root -p flush-hosts” to flush the host cache if some of the hosts change IP address or if the error message Host 'host_name' is blocked occurs

Related HA Command(s)

```
# ./ha_cntl --help
usage: ha_cntl [--activate cluster[.group[.instance]] | --deactivate
               cluster[.group[.instance]] | --diagnostics
               cluster[.group[.instance]]  [ --assumeyes ] | --view ] [
               --loglevel (INFO|WARN|ALL) ] [ --time_out <seconds> ] |
               --help

MoogSoft ha_cntl: Utility to send high availability control command
 -a,--activate <arg>      Activate all groups within a cluster, a specific
                          group within a cluster or a single instance
 -d,--deactivate <arg>    Deactivate all groups within a cluster, a
                          specific group within a cluster or a single
                          instance
 -h,--help                Print this message
 -i,--diagnostics <arg>   Print additional diagnostics where available to
                          process log file.
 -l,--loglevel <arg>      Specify (INFO|WARN|ALL) to choose the amount of
                          debug output
 -t,--time_out <arg>      Amount of time (in seconds) to wait for the last
                          answer (default 2 seconds)
 -v,--view                View the current status
 -y,--assumeyes           Answer yes for all questions
```

HA - Summary for Cisco Crosswork Situation Manager

1. Cisco Crosswork Situation Manager architecture: Nginx replaces httpd and Elasticsearch replaces Sphinx.
2. UI servlets all now act as one HA "instance". This is described in ticket MOOG-3825 v6 Release Notes.
3. The moogsvr hastatus endpoint still works as previously but the moogpoller hastatus endpoint has been removed (due to websockets changes). As the UI servlets all act as one instance now, only the moogsvr hastatus needs to be used to determine the UI's active /passive status. Additionally, a passive moogpoller servlet will not accept incoming websocket connections and going passive (from active) will disconnect existing websocket connections.

4. The switch to websockets means that care must be taken with any load balancers fronting an active/passive UI pair to ensure that the websocket connection does not get periodically reset. If this happens this will cause the UI to refresh by itself. An example config for haproxy is below that I will be using in the HA docs for a load balancer on qatest1 that is fronting an HA pair of UI's on qatest2 and qatest3. It is the "timeout client" and "timeout server" settings here (at their maximum allowed values of 24days) that prevent the websocket being disconnected:

```
global
        log 127.0.0.1   local2
        log 127.0.0.1   local1 notice
        maxconn 4096
        chroot /var/lib/haproxy
        user haproxy
        group haproxy
        daemon
        #debug
        #quiet


defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        option redispatch
        retries 3
        maxconn 2000
timeout connect 5000
   timeout client 24d
timeout server 24d


listen haproxy-monitoring *:9090
  mode    http
  stats   enable
  stats   show-legends
  stats   refresh           5s
  stats   uri               /
  stats   realm             Haproxy\ Statistics
  stats   auth              admin:admin
  stats   admin             if TRUE


frontend ui_front
  bind qatest1:443
  option tcplog
  mode tcp
  default_backend ui_back

backend ui_back
  mode tcp
  balance source
  option httpchk GET /moogsvr/hastatus
  http-check expect status 204
```

```
server ui_1 qatest2:443 check check-ssl verify none inter 100
server ui_2 qatest3:443 check check-ssl verify none inter 100
```

5) The "failover.margin" property in system.conf has been increased from 3 to 10 seconds. This only comes into play if moog_farmd auto-failover is being used and will mean that the passive moog_farmd will allow a little more time before attempting to take over processing. This therefore makes auto-failover a little less "twitchy". Auto-failover (for moog_farmd) was introduced in 5.2.0 and explained there: https://docs.moogsoft.com/display/050200/Incident.MOOG+5.2.0

HA - Setup for Dependencies

You can configure Cisco Crosswork Situation Manager dependencies such as Apache Tomcat, Nginx, MySQL, Grafana, RabbitMQ and Elasticsearch to work effectively in highly available deployments.

See High Availability (HA) for details on high availability deployments of Cisco Crosswork Situation Manager and deployment scenarios.

Configure Apache Tomcat for HA

You can set up Apache Tomcat for high availability using multiple Tomcat servers that you can cluster.

See HA - Deployment Scenarios and HA - Reference Architectures for examples. Refer to Apache's documentation about Tomcat Clustering.

Configure Nginx for HA

You can configure Nginx to run on multiple servers in a highavailability cluster to remove any single points of failure in your distributed deployment.

See HA - Deployment Scenarios and HA - Reference Architectures for examples. Refer to Nginx's documentation about High Availability.

Configure MySQL for HA

You can configure MySQL servers in a master-slave setup to ensure your distributed deployment of Cisco Crosswork Situation Manager is highly available. See HA - Deployment Scenarios and HA - Reference Architectures for examples. Refer to MySQL's documentation about High Availability.

Configure RabbitMQ for HA

You can improve the performance and reliability of your Cisco Crosswork Situation Manager

- deployment by: Distributing your RabbitMQ brokers on different hosts.
- Clustering your multiple RabbitMQ brokers.
- Mirror your message queues across multiple nodes.

See Message System Deployment for setup steps. Refer to the RabbitMQ documentation on Clustering and Mirrored Queues.

Configure Elasticsearch for HA

There are different ways to configure Elasticsearch for distributed installations. See HA - Deployment Scenarios for more information.

Refer to the Elasticsearch documentation about Clustering for more details.

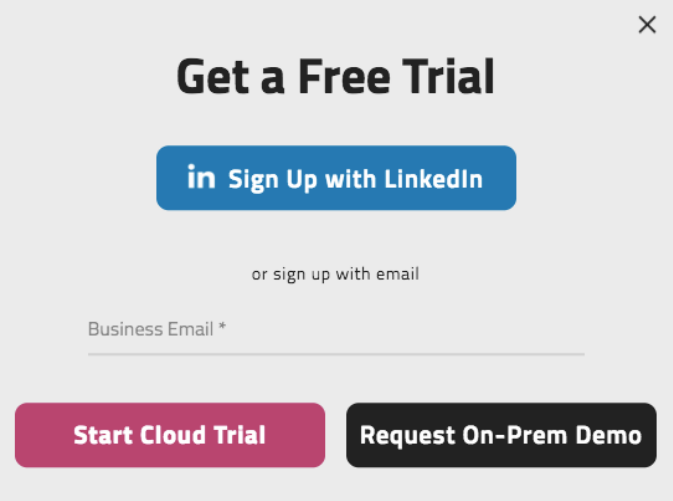
Configure Grafana for HA

To configure Grafana for distributed installations, you should configure each Grafana instance to connect to a Cisco Crosswork Situation Manager UI load balancer such as HAProxy rather than the Cisco Crosswork Situation Manager UI stack.

Alternatively you can point it at the Apache Tomcat server or Nginx server. Refer to the Grafana documention on Setting Up Grafana for High Availability.

Cisco Crosswork Situation Manager Trial

Cisco Crosswork Situation Manager is available for trial either on the Cisco Cloud or On-Prem.



Cloud

Enter your email address and click Start Cloud Trial. We'll send you details of how to get started with Cisco Crosswork Situation Manager on the Cisco Cloud.

On-Prem

Enter your email address and click Request On-Prem Demo. We'll send you details of how to get started with Cisco Crosswork Situation Manager On-Prem. You need to install the Docker Image Engine and then install the Cisco Docker Image.

Install the Docker Engine

Download the Docker Community Edition for your platform - it's available for all platforms.

Install the Cisco Crosswork Situation Manager Trial

1. Open the Command line interface.
2. Create a text file, **config.json** and save it to a folder named **.docker** in your home directory. E.g. **/Users/Yourname/.docker/config.json**
   This file contains the username and password you need to download the Cisco Trial Image and the content is provided by Cisco when you request an On-Prem Trial.
   Here's an example:

```
{
    "auths": {
    "https://index.docker.io/v1/": {
        "auth": "ZGNhcb&****NOTREALyoCCCGFbmU=",
        "email": "dave@moogsoft.com"
        }
    }
}
```

3. Pull the Cisco Crosswork Situation Manager docker image:

```
docker pull moog/saas:latest
```

4. Run the Cisco Crosswork Situation Manager docker image:

```
docker run -it -d -p 443:443 moog/saas
```

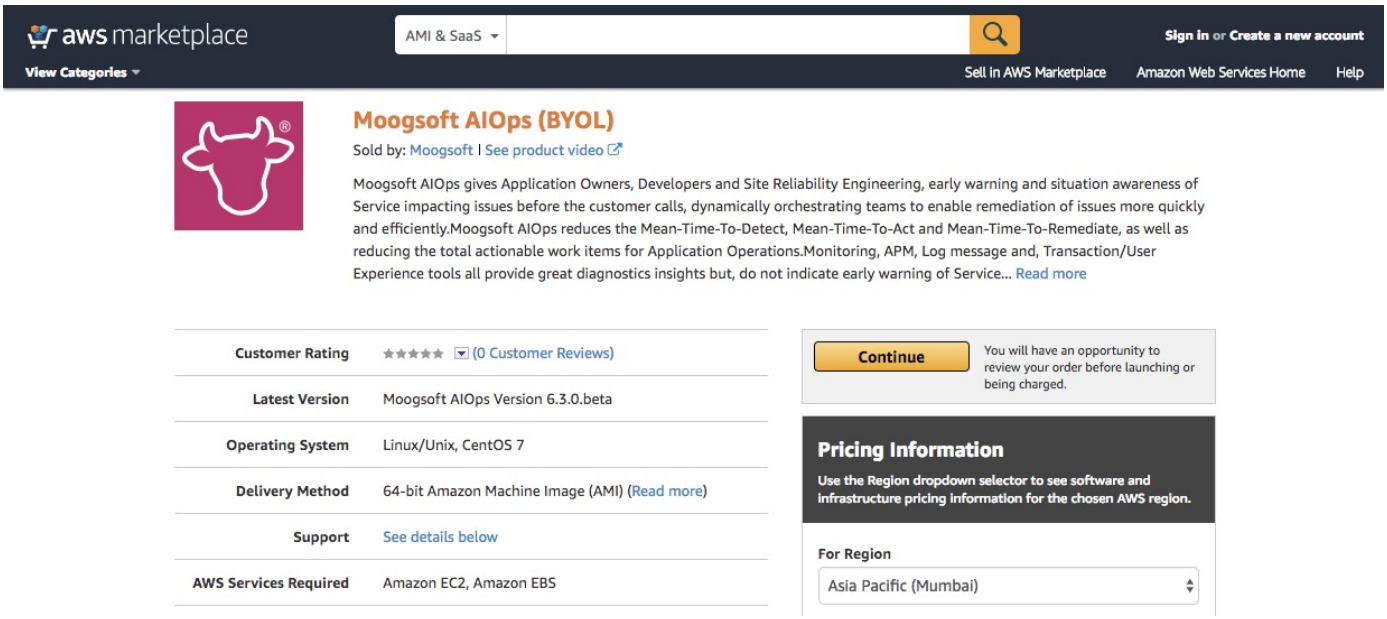
5. Open a browser: https://localhost to see the Cisco Crosswork Situation Manager UI.

> Ignore any error message from your browser i.e. Self-signed Certificates. There are no security issues since your are running Cisco Crosswork Situation Manager locally

6. Sign in with username **admin** and password **admin.** You are now running the Cisco Crosswork Situation Manager trial. Follow the in-app tutorials to learn how to use Cisco Crosswork Situation Manager.

AWS Marketplace Installation

You can install an instance of the latest version of Cisco Crosswork Situation Manager from the Amazon Web Services (AWS) Marketplace.



For more information see the AWS Marketplace listing.

Load Data into AWS Instance

You can start loading data into your AWS Marketplace instance by configuring SSL and setting a password for the limited user account.

Configure SSL

Cisco Crosswork Situation Manager includes a self-signed certificate by default. To remove the browser warning, you can add your own certificate as follows:

1 Replace the contents of `/etc/nginx/ssl/certificate.pem` and `/etc/nginx/ssl/certificate.key` with a self-signed or valid certificate.
2 Restart Nginx (`` `systemctl restart nginx` ``) or alternatively offload the TLS certificate to an AWS application load balancer.

> To SSH into AWS instance, the default password is `centos`.

Set Password for Tool Runner

Many of the Cisco Crosswork Situation Manager integrations run as a limited user account for security reasons. The user account, `moogtoolrunner` ,needs password before any data is loaded and the services are restarted. For more information see Tool Runner.

The entire step can be achieved using the following command in an SSH terminal:

```
export PASSWORD=<set a password here>; bash -c "$(curl https://s3.moogsoft.com/downloads/moogsetup.sh)"
```

If changing the password was successful, the following message appears:

```
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1170  100  1170    0     0   1105      0  0:00:01  0:00:01 --:--:--  1106
Changing password for user moogtoolrunner.
passwd: all authentication tokens updated successfully.
Match User moogtoolrunner
PasswordAuthentication yes
```

Please note: The default username for your AWS instance is 'Admin' and the password is your ECS instance ID, e.g. "i-0dd3a563981f57341". These can be changed or you can create a new login when you first sign in.

Encrypt Database Communications

You can enable SSL to encrypt communications between all Cisco Crosswork Situation Manager components and the MySQL database.

For information on creating SSL keys and certificates for MySQL, see Creating SSL and RSA Certificates and Keys using MySQL.

Establish Trust for the MySQL Certificate

To establish trust for the MySQL database certificate, create a truststore to house the root certificate for the Certificate Authority that signed the MySQL Server certificate.

1. If you upgraded from a previous version of Cisco Crosswork Situation Manager, run the following command to extract the certificate for the root CA for MySQL:

   ```
   mysql_ssl_rsa_setup
   ```

   The command generates new keys and writes them to the `/var/lib/mysql` directory.
2. Run the java `keytool` command to create a trust store containing the certificate for the root CA for MySQL.

   ```
   keytool -import -alias mysqlServerCACert -file /var/lib/mysql/ca.pem -keystore $MOOGSOFT_HOME/etc/truststore
   ```

   - When `keytool` prompts you, enter a password for the keystore. You will need this password to configure Cisco
   - Crosswork Situation Manager. Answer 'yes' to "Trust this certificate."

   Keytool creates a truststore at the path `$MOOGSOFT_HOME/etc/truststore`.

Configure Cisco Crosswork Situation Manager to use SSL for Database Communications

After you have created the truststore, edit the Cisco Crosswork Situation Manager configuration to enable SSL.

1. Edit `$MOOGSOFT_HOME/config/system.conf`.
2. Inside the MySQL property, uncomment the SSL property and the properties that comprise it. Make sure to uncomment the opening "`{`" and closing braces "`}`". For example:

```
,“ssl” :
{
# # The location of the SSL truststore.
# #
# # Relative pathing can be used, i.e. ‘.’ to mean current directory,
# # ‘../truststore’ or ‘../../truststore’ etc. If neither relative
# # nor absolute (using ‘/’) path is used then $MOOGSOFT_HOME is
# # prepended to it.
# # i.e. “config/truststore” becomes “$MOOGSOFT_HOME/config
/truststore”
# #
# #
# # Specify the server certificate.
# #
“trustStorePath” : “etc/truststore”,

# “trustStoreEncryptedPassword” : “vQj7/yom7e5ensSEb10v2Rb/pgkaPK
/4OcUlEjYNtQU=“,

“trustStorePassword” : “moogsoft”
}
```

3. Provide the path to the truststore you created. For example:

   ```
   "trustStorePath" : "etc/truststore",
   ```

4. Edit the password for the truststore. For example:

   ```
   "trustStorePassword" : "moogsoft"
   ```

   See Moog Encryptor if you want to use an encrypted password. Uncomment `trustStoreEncryptedPassword` and provide the encrypted password for the value. For example:

   ```
   “trustStoreEncryptedPassword” : “vQj7/yom7e5ensSEb10v2Rb/pgkaPK
   /4OcUlEjYNtQU=“
   ```

5. Save your changes and restart the following components:

   - moogfarmd
   - Apache Tomcat
   - all LAMs

After you restart, all Cisco Crosswork Situation Manager components encrypt communications with the MySQL database.

Scale Your Cisco Crosswork Situation Manager Implementation

Cisco Crosswork Situation Manager supports several options to help you scale your implementation to meet your performance needs. Use the built-in diagnostic tools to monitor your system for signs that it is time to scale.

For information on the performance tuning capabilities of individual Cisco Crosswork Situation Manager components, see Cisco Crosswork Situation Manager Component Performance.

Horizontal Scaling

Cisco Crosswork Situation Manager currently supports horizontal scaling at the integration (LAM) and visualization (Ngnix + Tomcat) layers.

- You can add more LAMs, either on additional servers or on the same server, to achieve higher event rates. In this case, you have the option to configure event sources to send to the parallel LAMs separately or to implement a load balancer in front of the LAMs.
- You can add Nginx/Tomcat UI "stacks" behind a load balancer to increase performance for UI users. Adding UI stacks does not always provide better performance. It can degrade performance by adding more connection pressure to the database.

The following are typical horizontal scaling scenarios:

- You can add an additional LAM to process incoming events if you see that, despite attempts to tune the number of threads for an individual LAM, its event rate hits a plateau. This is a sign that the LAM is the bottleneck, so adding other instances of the LAM behind a load balancer will allow a higher event processing rate.
- You can add an additional UI stack if database pool diagnostics for Tomcat suggest that all or most of the database connections are constantly busy with long running connections, but the database itself is performing fine.

The data processing layer (moogfarmd) is not currently well suited to horizontal scaling. Moolets of the same type cannot currently share processing. Adding more Moolets like the AlertBuilder in an attempt to increase the event processing rate is likely to lead to database problems.

Vertical Scaling

All Cisco Crosswork Situation Manager components ultimately benefit from being run on the best available hardware, but the data processing layer (moogfrarmd) benefits most from this approach. Depending on the number and complexity of Moolets in your configuration, you will see performance benefits in data processing on servers having the fastest CPUs with numerous cores and a large amount of memory. This enables you to increase the number of threads for moogfarmd to improve processing speed. You should also locate the database on the most feasibly powerful server (clock speed, number of cores and memory) with the biggest/fastest disk.

Distributed Installations

In some cases you distribute Cisco Crosswork Situation Manager components among different hosts to gain performance because it reduces resource contention on a single server: The most common distribution is to install the database on a separate server, ideally within the same fast network to minimize risk of latency. An additional benefit of this move is that it allows you to run a clustered or master/slave database for redundancy.

Another common distribution is to install the UI stack (Nnginx) on a separate server within the same fast network.

Some integrations (LAMs) benefit in being closer to the source so are a candidates for distribution. You can also break the data processing layer (moogfarmd) into its component Moolets and installed across separate servers. This gives each moogfarmd process a separate database pool which may provide performance benefits in certain situations.

See Distributed Installation - RPM for more information.

Cisco Crosswork Situation Manager Component Performance

Cisco Crosswork Situation Manager features the abilities to ingest large amounts of event data from various sources, process the data using configurable logic, and to display the data to multiple concurrent users.. This document outlines the various system components and how their interactions can impact system performance. It includes performance tuning suggestions where applicable.

To learn about opportunities to plan your implementation for increased performance capabilities, see Scaling Your Cisco Crosswork Situation Manager Implementation.

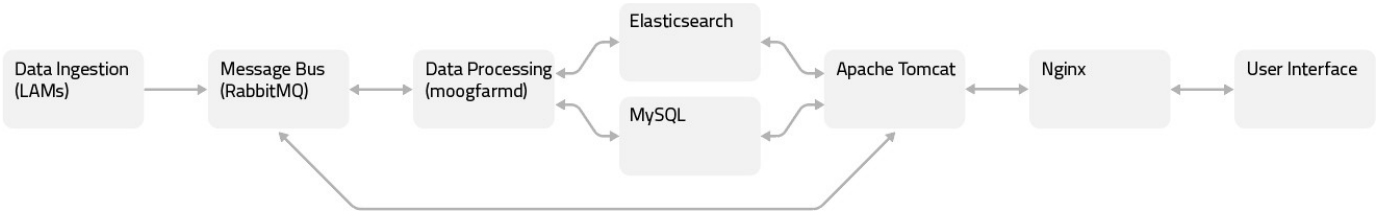
For information on monitoring your system performance and handling performance issues, see Monitor and Troubleshoot Cisco Crosswork Situation Manager.

Sytem Component Summary

Cisco Crosswork Situation Manager comprises several components which have tuning and configuration options available:

- Integrations (LAMs) that listen or poll for events, parse and encode them into discrete events, and then write the discrete events to the message bus.
- The message bus (RabbitMQ) that receives published messages from integrations. It publishes messages destined for data processing (moogfarmd) and the web application server.
- The system datastore (MySQL) that handles transactional data from other parts of the system: integrations (LAMs), data processing, and the web application server.
- The data processing component (moogfarmd), an application that consumes messages from the message bus. It processes event data in a series of servlet-like modules called Moolets. Moogfarmd reads and writes to the database and publishes messages to the bus.
- The web application server (Tomcat) that reads and writes to the bus and the database.

The diagram below shows the general data flow for the components:



Other components include:

- A proxy (Ngnix) for the web application server and for integrations. See the Ngnix docs for more information.
- The search engine (Elasticsearch) for the UI that indexes documents from the indexer moolet in the data processing series. It returns search results to Tomcat. See the Elasticsearch docs for more information.

Integration Performance

Event data enters the system via integrations (LAMs). Integrations running on a large system can normally process up to 10,000 events per second and publish them to the message bus at an equivalent rate. Integrations can buffer events under event storm conditions. The following factors affect an integration's capacity to process events:

- CPU clock speed and number of available cores.
- Threads setting.
- Complexity of the LAMbot logic.
- Whether you have enabled "guaranteed delivery settings". For example, `rest_response_mode` for the REST LAM.
- Log level. For example, DEBUG is the slowest.

You can specify a value for `num_threads` in the Integration configuration file to control the number of active threads for the integration. To tune the socket LAM, for example, edit `socket_lam.conf`. Increasing the number of threads can improve ingestion performance. However it will also result in higher CPU usage and may cause internal buffering in the integration. Buffering increases memory usage until the buffer is cleared.

Message Bus Performance

RabbitMQ is very lightweight and, in all known cases, has been able to process the incoming event rate from Integrations. Refer to the RabbitMQ documentation for its performance tuning options.

Database Performance

You should manage and tune your MySQL instance as you would any other database system in your enterprise. In addition to the standard tuning options in the MySQL documentation, consider the following recommendations for settings in `/etc/my.cnf`:

- On servers with >= 16 GB RAM that run MySQL and Cisco applications ,set `innodb-buffer-pool-size` to 50% of system RAM. On
- servers where only MySQL is running, set `innodb-buffer-pool-size` to 80% of system RAM.
- If `innodb-buffer-pool-size` > 8 GB, increase the `innodb-buffer-pool-instances` to divide the buffer buffer pool into 1 G (GB) chunks to the maximum supported value of 64 G. For example, if your buffer pool size is 64 GB:

```
innodb-buffer-pool-size=64G
innodb_buffer_pool_instances=64
```

Data Processing Performance

The data processing component for Cisco Crosswork Situation Manager, moogfarmd, is the most complex and configurable system component. It offers a range of performance capabilities depending on which Moolets you use and the workload for those Moolets. The following factors affect moogfarmd performance:

- Incoming event rate from Integrations (LAMs).
- CPU clock speed and number of available cores.
- Available memory and `-Xmx` setting of moogfarmd process.
- Top-level and per-moolet threads setting.
- Number of Moolets and their complexity and/or interaction with external services.
- Database load from other parts of the system, for example API requests.
- Incoming messages from the bus or other parts of the system.

Moogfarmd buffers messages in message storm conditions. Each Moolet has its own message queue that enables it to handle message storms and process the backlog once the storm has cleared.

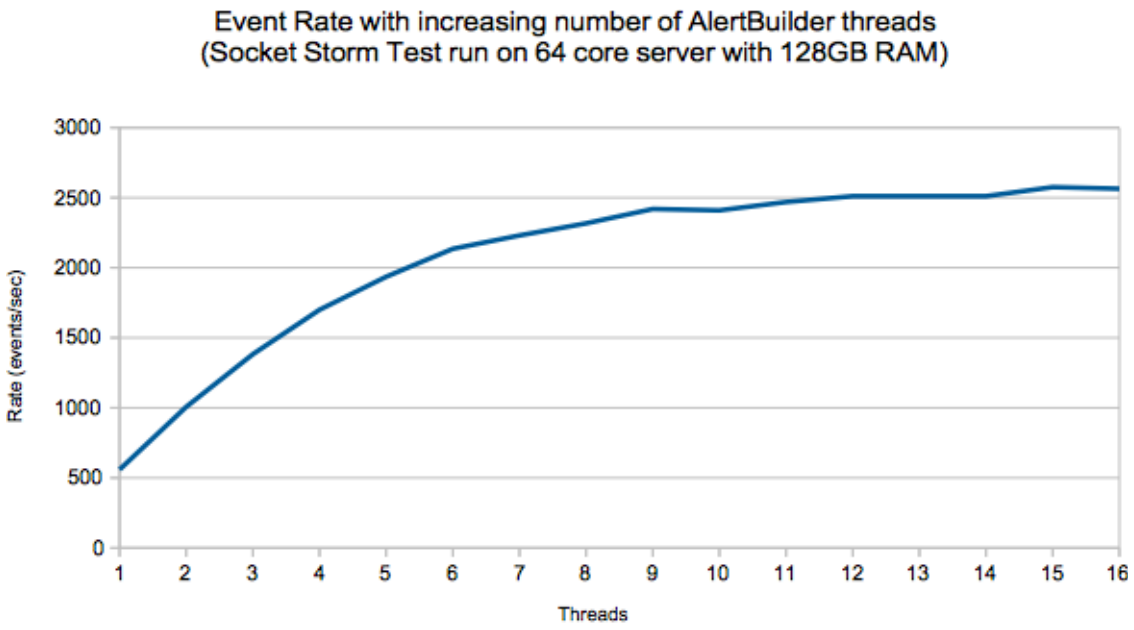
You can configure thread allocation for moogfarmd in the `$MOOGSOFT_HOME/config/moog_farmd.conf` file as follows:

- The top-level threads property controls the following:
    - The default number of threads for each Moolet unless you specify a setting for a particular Moolet.
    - The size of the database pool for moogfarmd to database connections.
- The per-Moolet level threads property allows individual control of the number of threads for a particular Moolet.

Increasing either setting can lead to improved processing performance but will likely increase CPU and memory usage. Too many threads can lead to overload of connections or transactions to the database and impact other areas of the system. For example, increasing the number of threads for the AlertBuilder Moolet can improve the event processing rate, but increases load on the database potentially causing deadlocks.

AlertBuilder Moolet

The main performance gateway for moogfarmd is the AlertBuilder because it interacts with MySQL the most. In simple configurations with a tuned MySQL database and no other load, you can increase number of threads for the AlertBuilder to process up to 2500 events per second and write them to the database at an equivalent rate. The following graph illustrates the performance impact of adding AlertBuilder threads for moogfarmd running only with AlertBuilder.



This scenario does not account for other Database load, other Moolets, or any custom logic added to the AlertBuilder Moobot. Event processing would run at about half this rate in a real-world case.

Sigalisers

Cisco Crosswork Situation Manager clustering algorithms or Sigalisers, employ complex calculations. Depending on its settings, the Sigaliser can account for a lot of processing time and memory within moogfarmd. It is impossible to predict a processing rate for these algorithms because as they vary greatly according to configuration and workload. Normally Sigalisers do not add much load to the database except in a burst of Situation creation.
Moogfarmd retains previously created active Situations in memory according to the `retention_period` setting in moog_farmd.conf. You can expect memory to grow in moogfarmd as a consequence under a high rate of Situation generation.

Other Moolets

The performance of other Moolets vary based upon configuration and the rate at which they receive messages to process. Moolets that interact with external services may introduce processing delay to moogfarmd when there is network or processing latency associated with the external service.

Web Application Server Performance

The Tomcat servlets provide the backend for the Cisco Crosswork Situation Manager UI which drives the end-user experience. Scalability tests show that a single Tomcat instance can support up to 500 concurrent UI users before response times degrade. Tomcat performance depends on the following factors:

- Incoming event rate from integrations (LAMs).
- Incoming messages from other parts of the system, such as moogfarmd.
- CPU clock speed and number of available cores.
- Available memory and -Xmx setting of Tomcat process.
- Database load from other parts of the system.
- Number and complexity of alert and Situation filters being used.
- Activities of the users.

To provide quicker load times for users, the UI employs caching benefits for filtered views. Tomcat writes to the message bus to cope with event or update storms.

The `db_connections` and `priority_db_connections` settings `$MOOGSOFT_HOME/config/servlets.conf` control the size of the database connection pool that Tomcat uses to connect to MySQL. You can increase either setting to potentially improve UI performance. Exercise caution when changing these values because increases to will typically increase CPU and memory usage of both the Tomcat and database processes. Too many connections can lead to an overload of transactions to the database which impacts other areas of the system.

JVM Performance

Integrations (LAMs), moogfarmd, and Tomcat are all java processes so you can tune the memory allocation pool settings for the JVM to optimize performance. This `-Xmx` setting defines the maximum allowed Java heap size of the process. The default memory allocation for a Java process is one quarter of the server's RAM.

For integrations and moogfarmd, you can add the -Xmx argument to the line in `$MOOGSOFT_HOME/bin/<lam name>` or `$MOOGSOFT_HOME/bin/moogfarmd` where the JVM is launched.

For example, to set the maximum Java heap size for the moogfarmd process to 16 GB, add "`-Xmx16g`" to the `java_vm` command line in `$MOOGSOFT_HOME/bin/moogfarmd` as follows:

```
#
# Run app
#
$java_vm -server -Xmx16g -DprocName=$proc_name -DMOOGSOFT_HOME=$MOOGSOFT_HOME -classpath
$java_classpath $java_main_class "$@" &
```

For Tomcat, the default setting is 2GB. If you need to change it you can edit the service script `/etc/init.d/apache-tomcat`.

Monitor and Troubleshoot Cisco Crosswork Situation Manager

This document details the available health and performance indicators included with the Cisco Crosswork Situation Manager system. It also provides some guidance on how to monitor your system and how to troubleshoot performance problems.

Processing Metrics

Navigate to **System Settings > Self Monitoring > Processing Metrics** to see a breakdown of the current state of the system based on the metrics received from the running components.

- The moogfarmd process and all LAMs publish detailed performance information.
- A bullet chart at the top of the page shows the key performance metric for the system: Current Maximum Event Processing Time. The defined performance ranges are color coded: good (green), marginal (yellow) and bad (red). As the metric changes the bullet chart updates to reflect good, marginal or bad performance.
- The system calclulates Current Maximum Event Processing Time as the approximate 95th percentile of the current maximum time in seconds that it takes for an event to make its way through the system from its arrival at a LAM until its final processing by a Moolet in moogfarmd.
- By default, AlertBuilder, AlertRulesEngine and All Sigalisers are used to calculate the Current Maximum Event Processing Time metric.
- You can configure the `metric_path_moolet` property in moog_farmd.conf to specify the Moolets to use to calculate Current Maximum Event Processing Time.
- By default, the good, marginal and bad ranges of the bullet chart are set to 0-10secs, 10-15secs and 15-20secs respectively. You can change the configuration in the in the `eventProcessingTimes` section in the `portal` block of `$MOOGSOFT_HOME/ui/html/web.conf`.

Good performance means LAMs are consuming and publishing events without problem as indicated by:

- Message Queue Size is 0.
- Socket Backlog (if relevant) is not increasing.

Additionally, moogfarmd is consuming and processing events successfully as indicated by all of:

- Total Abandoned Messages is 0 for the majority of the time.
- Asynchronous Task Queue Size is 0 for the majority of the time.
- Cookbook Resolution Queue is 0 for the majority of the time.
- Message Backlogs for all Moolets is 0 for the majority of the time.
- The Messages Processed count for all running moolets should be the same (unless custom configuration causes event routing through different moolets) i.e. no moolet is falling behind.

The above should lead to a stable low Current Maximum Event Processing Time depending on the complexity of the system.

Marginal or **Bad** performance means LAMs are not consuming and publishing events at the rate at which they receive them, as indicated by:

- Message Queue Size is > 0 and likely increasing.
- Socket Backlog is increasing.

Additionally, moogfarmd is not consuming and processing events in a timely fashion as indicated by some or all of:

- Total Abandoned Messages is constantly > 0 and likely increasing.
- Asynchronous Task Queue Size is > 0 and likely increasing.
- Cookbook Resolution Queue is constantly > 0 and likely increasing.
- Message Backlogs for all Moolets is constantly > 0 and likely increasing.
- The Messages Processed count for all running Moolets is not the same indicating that some Moolets are falling behind. This doesn't apply for cases where custom configuration causes event routing through different Moolets.

The above will likely lead to an unstable high Current Maximum Event Processing Time depending on the complexity of the system.

See Self Monitoring for more detail.

Graze getSystemStatus Endpoint

The getSystemStatus endpoint returns useful information about running processes within the system. For example:

```
curl -u graze:graze -k "https://localhost/graze/v1/getSystemStatus"
```

See Graze API for more detail.

moogfarmd Health Logging

moogfarmd writes detailed health information in JSON format to its log file once a minute. Information falls into five logical blocks:

- totals: running totals since moogfarmd was started.
- interval_totals: running totals since the last 60 second interval)
- current_state: a snapshot of the important queues in moogfarmd
- garbage_collection: JVM garbage collection data
- JVM_memory: JVM memory usage data

Example output:

```
WARN : [HLog   ][20180510 20:39:55.538 +0100] [CFarmdHealth.java]:533 +|
{"garbage_collection":{"total_collections_time":12827,"
last_minute_collections":0,"last_minute_collections_time":0,"
total_collections":1244},"current_state":{"pending_changed_situations":0,"
total_in_memory_situations":4764,"situations_for_resolution":0,"
event_processing_metric":0.04747474747474475,"message_queues":
{"AlertBuilder":0,"TeamsMgr":0,"Housekeeper":0,"Indexer":0,"
bus_thread_pool":0,"Cookbook3":0,"Cookbook1":0,"SituationMgr":0,"
SituationRootCause":0,"Cookbook2":0},"in_memory_entropies":452283,"
cookbook_resolution_queue":0,"total_in_memory_priority_situations":0,"
```

```
created_priority_situations":0,"created_external_situations":0,"
created_situations":10,"messages_processed":{"TeamsMgr":182,"Housekeeper":
0,"AlertBuilder":1782,"Indexer":2082,"Cookbook3":1782,"SituationRootCause":
172,"Cookbook1":1782,"SituationMgr":172,"Cookbook2":1782},"
alerts_added_to_priority_situations":0,"alerts_added_to_situations":111,"
situation_db_update_failure":0},"JVM_memory":{"heap_used":1843627096,"
heap_committed":3007840256,"heap_init":2113929216,"nonheap_committed":
66912256,"heap_max":28631367680,"nonheap_init":2555904,"nonheap_used":
64159032,"nonheap_max":-1},"totals":{"created_events":453252,"
created_priority_situations":0,"created_external_situations":0,"
created_situations":4764,"alerts_added_to_priority_situations":0,"
alerts_added_to_situations":36020,"situation_db_update_failure":0}}|+
```

In a healthy system that is processing data:

- The count of created events and created Situations should increase.
- The messages_processed should show that Moolets are processing messages.
- The current_state.message_queues should not be accumulating (there may be spikes).
- The total_in_memory Situations should increase over time but will reduce periodically due to the retention_period.
- The situation_db_update_failure should be zero.

Tomcat Servlet Logging

Tomcat writes counter information from each of the main servlets to its `catalina.out` once a minute.

Example output:

```
WARN : [Thread-][20180510 20:57:05.501 +0100] [CReporterThread.java]:136
+|MoogPoller read [16722] MooMs messages in the last [60] seconds.|+
WARN : [Thread-][20180510 20:57:07.169 +0100] [CReporterThread.java]:136
+|Graze handled [55] requests in the last [60] seconds.|+
WARN : [Thread-][20180510 20:57:10.181 +0100] [CReporterThread.java]:136
+|MoogSvr handled [86] requests in the last [60] seconds.|+
WARN : [Thread-][20180510 20:58:03.197 +0100] [CReporterThread.java]:136
+|Situations similarity component calculated similarities for [264]
situations in the last [60] seconds.|+
```

The counters are:

- Number of MoogSvr requests in the last minute (i.e. number of standard UI requests made).
- Number of Moogpoller MooMs messages in the last minute (i.e. number of messages read from the bus).
- Number of Graze requests in the last minute.
- Number of similar Situations calculated in the last minute.

In a healthy system that is processing data:

- The Moogpoller count should always be non-zero.
- The MoogSvr and Graze counters may be zero, but should reflect the amount of UI and Graze activity.
- The similar Situations counter may be zero but should reflect the number of similar Situations that are occurring in the system.

Database Pool Diagnostics

Cisco Crosswork Situation Manager features a capability to print out the current state of the DBPool in both moogfarmd and Tomcat. This can be very useful to diagnose problems with slow event processing or UI response.

To trigger logging, run the `ha_cntl` utility and pass the cluster name using the -i argument. For example:

```
ha_cntl -i MOO
This will perform task "diagnostics" all groups within the [MOO] cluster.
Diagnostics results will be in the target process log file.
Are you sure you want to continue? (y/N)
```

The utility triggers logging to `/var/log/moogsoft/moogfarmd.log`. For example in a performant system:

```
WARN : [pool-1-][20180511 10:06:07.690 +0100] [CDbPool.java]:792 +|[farmd]
DATABASE POOL DIAGNOSTICS:|+
WARN : [pool-1-][20180511 10:06:07.690 +0100] [CDbPool.java]:793 +|[farmd]
Pool created at [20180510 17:54:48.911 +0100].|+
WARN : [pool-1-][20180511 10:06:07.690 +0100] [CDbPool.java]:797 +|[farmd]
[2] invalid connections have been removed during the lifetime of the pool.
|+
WARN : [pool-1-][20180511 10:06:07.690 +0100] [CDbPool.java]:833 +|[farmd]
Pool size is [10] with [10] available connections and [0] busy.|+
```

It also triggers logging to `/usr/share/apache-tomcat/logs/catalina.out`. For example:

```
WARN : [0:MooMS][20180511 10:06:07.690 +0100] [CDbPool.java]:792 +|
[SituationSimilarity] DATABASE POOL DIAGNOSTICS:|+
WARN : [0:MooMS][20180511 10:06:07.690 +0100] [CDbPool.java]:793 +|
[SituationSimilarity] Pool created at [20180510 17:55:04.262 +0100].|+
WARN : [3:MooMS][20180511 10:06:07.690 +0100] [CDbPool.java]:792 +|
[MoogPoller] DATABASE POOL DIAGNOSTICS:|+
WARN : [3:MooMS][20180511 10:06:07.690 +0100] [CDbPool.java]:793 +|
[MoogPoller] Pool created at [20180510 17:55:01.990 +0100].|+
WARN : [0:MooMS][20180511 10:06:07.690 +0100] [CDbPool.java]:833 +|
[SituationSimilarity] Pool size is [5] with [5] available connections and
[0] busy.|+
WARN : [3:MooMS][20180511 10:06:07.691 +0100] [CDbPool.java]:833 +|
[MoogPoller] Pool size is [10] with [10] available connections and [0]
busy.|+
WARN : [1:MooMS][20180511 10:06:07.693 +0100] [CDbPool.java]:792 +|
[ToolRunner] DATABASE POOL DIAGNOSTICS:|+
WARN : [1:MooMS][20180511 10:06:07.694 +0100] [CDbPool.java]:793 +|
[ToolRunner] Pool created at [20180510 17:55:00.183 +0100].|+
WARN : [1:MooMS][20180511 10:06:07.694 +0100] [CDbPool.java]:792 +|
[MoogSvr : priority] DATABASE POOL DIAGNOSTICS:|+
WARN : [1:MooMS][20180511 10:06:07.694 +0100] [CDbPool.java]:833 +|
[ToolRunner] Pool size is [5] with [5] available connections and [0] busy.
|+
WARN : [1:MooMS][20180511 10:06:07.694 +0100] [CDbPool.java]:793 +|
[MoogSvr : priority] Pool created at [20180510 17:54:56.800 +0100].|+
WARN : [1:MooMS][20180511 10:06:07.695 +0100] [CDbPool.java]:797 +|
[MoogSvr : priority] [5] invalid connections have been removed during the
lifetime of the pool.|+
WARN : [1:MooMS][20180511 10:06:07.695 +0100] [CDbPool.java]:833 +|
[MoogSvr : priority] Pool size is [25] with [25] available connections and
[0] busy.|+
```

```
WARN : [1:MooMS][20180511 10:06:07.695 +0100] [CDbPool.java]:792 +|
[MoogSvr : normal priority] DATABASE POOL DIAGNOSTICS:|+
WARN : [1:MooMS][20180511 10:06:07.695 +0100] [CDbPool.java]:793 +|
[MoogSvr : normal priority] Pool created at [20180510 17:54:56.877 +0100].
|+
WARN : [1:MooMS][20180511 10:06:07.695 +0100] [CDbPool.java]:833 +|
[MoogSvr : normal priority] Pool size is [50] with [50] available
connections and [0] busy.|+
```

In both of these examples, the connections are "available" and none show as busy. However, in a busy system with flagging performance, moogfarmd.log will show different results. In the example below, all connections are busy and have been held for a long time. This type of critical issue causes moogfarmd to stop processing:

```
WARN : [pool-1-]20180309 16:49:30.031 +0000] [CDbPool.java]:827 +|[farmd]
Pool size is [10] with [0] available connections and [10] busy.|+
WARN : [pool-1-][20180309 16:49:30.031 +0000] [CDbPool.java]:831 +|The
busy connections are as follows:
1: Held by 5:SituationMgrLOGFILECOOKBOOK for 173603 milliseconds. Checked
out at [CArchiveConfig.java]:283.
2: Held by 7:SituationMgrSYSLOGCOOKBOOK for 173574 milliseconds. Checked
out at [CArchiveConfig.java]:283.
3: Held by 8:SituationMgrSYSLOGCOOKBOOK for 173658 milliseconds. Checked
out at [CArchiveConfig.java]:283.
4: Held by 9:SituationMgrSYSLOGCOOKBOOK for 173477 milliseconds. Checked
out at [CArchiveConfig.java]:283.
5: Held by 8:TeamsMgr for 173614 milliseconds. Checked out at
[CArchiveConfig.java]:283.
6: Held by 4:SituationMgrSYSLOGCOOKBOOK for 173514 milliseconds. Checked
out at [CArchiveConfig.java]:283.
7: Held by 5:PRC Request Assign - SituationRootCause for 173485
milliseconds. Checked out at [CArchiveConfig.java]:283.
8: Held by 2:SituationMgrSYSLOGCOOKBOOK for 173661 milliseconds. Checked
out at [CArchiveConfig.java]:283.
9: Held by 6:SituationMgrSYSLOGCOOKBOOK for 173631 milliseconds. Checked
out at [CArchiveConfig.java]:283.
10: Held by 6:TeamsMgr for 172661 milliseconds. Checked out at
[CArchiveConfig.java]:283.|+
```

It is expected that occasionally some of the connections will be busy but as long as they are not held for long periods of time then the system will be functioning normally.

You can use the following bash script to automatically gather DBPool diagnostics:

```
#!/bin/bash

#Get the cluster name
CLUSTER=$($MOOGSOFT_HOME/bin/utils/moog_config_reader -k ha.cluster)

#Get the current line numbers of latest log lines
FARMLINES=$(wc -l /var/log/moogsoft/moogfarmd.log|awk '{print $1}')
```

```
TOMLINES=$(wc -l /usr/share/apache-tomcat/logs/catalina.out|awk '{print
$1}')

#Run ha_cntl -i <cluster>
ha cntl -i $CLUSTER -y > /dev/null

sleep 5

#Print the results
echo "moog_farmd:"
tail -n +$FARMLINES /var/log/moogsoft/moogfarmd.log|egrep "CDbPool|Held by"

echo "tomcat:"
tail -n +$TOMLINES /usr/share/apache-tomcat/logs/catalina.out|egrep
"CDbPool|Held by"
```

To run the script, execute the following command:

```
./get_dbpool_diag.sh
```

MySQL Slow Query Logging

Slow query logging captures long running queries that are impacting the database. You can enable the feature as follows:

1. Check the current settings in MySQL for the feature:

```
mysql> show variables like '%slow_query_log%';
+---------------------+--------------------------+
| Variable_name       | Value                    |
+---------------------+--------------------------+
| slow_query_log      | OFF                      |
| slow_query_log_file | /var/log/mysql-slow.log |
+---------------------+--------------------------+
2 rows in set (0.00 sec)

mysql> show variables like 'long_query%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)
```

2. Ensure that the file specified in the slow_query_log_file setting exists and has the correct permissions. If not create it/set permissions :

```
touch /var/log/mysql-slow.log
chown mysql:mysql /var/log/mysql-slow.log
```

3. Enable the slow query log:

```
mysql> set global slow_query_log=on;
```

After that, queries that take longer than 10 seconds to execute will appear in the log file. For example:

```
/usr/sbin/mysqld, Version: 5.7.19 (MySQL Community Server (GPL)).
started with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
Time Id Command Argument
# Time: 2018-02-02T19:12:20.822756Z
# User@Host: ermintrude[ermintrude] @ localhost [127.0.0.1] Id: 98
# Query_time: 55.161516 Lock_time: 0.000025 Rows_sent: 1
Rows_examined: 25878591
use moogdb;
SET timestamp=1517598740;
SELECT COALESCE(MIN(GREATEST(last_state_change,last_event_time)),
UNIX_TIMESTAMP(SYSDATE())) as oldest FROM alerts WHERE alerts.
alert_id NOT IN (SELECT sig_alerts.alert_id FROM sig_alerts);
# Time: 2018-02-02T19:13:19.255277Z
# User@Host: ermintrude[ermintrude] @ localhost [127.0.0.1] Id: 98
# Query_time: 56.131108 Lock_time: 0.000028 Rows_sent: 515
Rows_examined: 25878591
SET timestamp=1517598799;
SELECT alerts.alert_id FROM alerts WHERE alerts.alert_id NOT IN
(SELECT sig_alerts.alert_id FROM sig_alerts) AND GREATEST
(last_state_change,last_event_time) BETWEEN 1417547486 AND 1417633885;
```

The value of the `long_query_time` setting can also be adjusted up or down as suits.

RabbitMQ Admin UI

RabbitMQ includes an admin UI that gives performance information about the message bus. By default this is accessible via http://<hostname>:15672 with credentials moogsoft/m00gs0ft. Check for the following scenarios:

- Early warning of any system resource issues in the "Nodes" section on the Overview page. For example, File/Socket Descriptors, Erlang Processes, Memory and Disk Space.
- Build-up of "Ready" messages in a queue - this indicates a message queue is forming. This means that the associated Cisco Crosswork Situation Manager process is not consuming messages from this queue. It could also point to an orphaned queue that no longer has an associated consumer. This could happen if "message_persistence" has been enabled in system.conf and moogfarmd and or Tomcat has been reconfigured with a different HA process group name.

See the RabbitMQ docs for information on how to use the admin UI.

Other Utilities

MySQLTuner provides useful diagnostics and recommendations on MySQL settings. See MySQLTuner for more information.

To monitor the CPU and Memory usage of the running components of a Cisco Crosswork Situation Manager system, you can run the following script that offers simple CPU and Memory monitoring of the RabbitMQ, Socket LAM, Farmd, Tomcat and MySQL processes:

```
#!/bin/bash
```

```
SLEEPTIME=$1

f_return_metrics() {

        PROCPID=$1
        TOPOUTPUT=`top -p $PROCPID -n1 | tail -2 | head -1 |sed 's/[^ ]
\+\s\(.*\)/\1/g'`
        PROCICPU=`echo $TOPOUTPUT| awk '{print $8}'`
        if [ "$PROCICPU" == "S" ]; then PROCICPU=`echo $TOPOUTPUT| awk
'{print $9}'`;fi
        PROCPCPU=`ps -p $PROCPID -o pcpu|tail -1|awk '{print $1}'`
        PROCMEM=`ps -p $PROCPID -o rss|tail -1|awk '{print $1}'`
        echo $PROCICPU,$PROCPCPU,$PROCMEM

}

#Capture PIDs
RABBITPID=`ps -ef|grep beam|grep -v grep|awk '{print $2}'`
LAMPID=`ps -ef|grep socket_lam|grep java|grep -v grep|awk '{print $2}'`
MYSQLPID=`ps -ef|grep mysqld|grep -v mysqld_safe|grep -v grep|awk '{print
$2}'`
TOMCATPID=`ps -ef|grep tomcat|grep java|grep -v grep|awk '{print $2}'`
FARMDPID=`ps -ef|grep moog_farmd|grep java|grep -v grep|awk '{print $2}'`

echo "DATE,TIME,RABBITICPU(%),RABBITPCPU(%),RABBITRSS(Kb),LAMICPU(%),
LAMPCPU(%),LAMRSS(Kb),FARMDICPU(%),FARMDPCPU(%),FARMDRSS(Kb),TOMCATICPU(%),
TOMCATPCPU(%),TOMCATRSS(Kb),MYSQLICPU(%),MYSQLPCPU(%),MYSQLRSS(Kb)"

while [ true ]; do

  DATENOW=`date +"%m-%d-%y"`
  TIMENOW=`date +"%T"`

  RABBITMEAS=$(f_return_metrics $RABBITPID)
  LAMMEAS=$(f_return_metrics $LAMPID)
  FARMDMEAS=$(f_return_metrics $FARMDPID)
  TOMCATMEAS=$(f_return_metrics $TOMCATPID)
  MYSQLMEAS=$(f_return_metrics $MYSQLPID)
  TOMCATMEAS=$(f_return_metrics $TOMCATPID)

  echo
"$DATENOW,$TIMENOW,$RABBITMEAS,$LAMMEAS,$FARMDMEAS,$TOMCATMEAS,$MYSQLMEAS"

  sleep $SLEEPTIME

done
```

Example usage and output:

```
[root@ldev04 640]# ./perfmon.sh 5
DATE,TIME,RABBITICPU(%),RABBITPCPU(%),RABBITRSS(Kb),LAMICPU(%),LAMPCPU(%),
LAMRSS(Kb),FARMDICPU(%),FARMDPCPU(%),FARMDRSS(Kb),TOMCATICPU(%),TOMCATPCPU
```

```
(%),TOMCATRSS(Kb),MYSQLICPU(%),MYSQLPCPU(%),MYSQLRSS(Kb)
05-10-18,22:44:
26,28.0,8.5,203068,2.0,1.0,557092,20.0,13.5,2853408,4.0,2.1,5680584,28.0,17
.4,9657152
05-10-18,22:44:
34,14.0,8.5,183492,4.0,1.0,557092,16.0,13.5,2850484,0.0,2.1,5680584,33.9,17
.4,9657152
05-10-18,22:44:
43,0.0,8.5,181072,0.0,1.0,557092,0.0,13.5,2850484,0.0,2.1,5680584,4.0,17.4,
9658312
05-10-18,22:44:
51,12.0,8.5,181040,0.0,1.0,557092,0.0,13.5,2850484,0.0,2.1,5680584,4.0,17.4
,9658312
05-10-18,22:44:
59,0.0,8.5,181040,0.0,1.0,557092,0.0,13.4,2850484,0.0,2.1,5680584,0.0,17.4,
9658312
```

Notes:

- Script only outputs to the console so should be redirected to a file for logging results
- Output is in csv format.
- ICPU = "Instantaneous CPU Usage (%)"
- PCPU = "Percentage of CPU usage since process startup (%)"
- RSS = "Resident Set Size i.e. Memory Usage in Kb"
- For CPU measurements a measure of 100% represents all of one processor so results > 100% are achievable for multi-threaded processes.

Troubleshooting Performance Problems

If the system is showing signs of latency in alert or Situation creation then the problem is likely with moogfarmd and/or the database**.** The following diagnostic steps will help you track down the cause:

Step	Description	Possible Cause and Resolution
1	Check moogfarmd.log for any obvious errors or warning.	Cause may be evident from any warnings or errors.
2	Check the Self Monitoring Processing Metrics Page	If the event_process_metric is large and/or increasing then something is backing up. Check moogfarmd health logging also for sign of message_queue build-up in any of the Moolets.
3	Check the CPU/Memory usage of the server itself.	If the server, as a whole, is running close to CPU or Memory limit and no other issues can be found (e.g. rogue processes or memory leaks in the Cisco Crosswork Situation Manager components) then consider adding more resource to the server or distributing the Cisco Crosswork Situation Manager components.
4	Check whether moog_farmd java process is showing constant high CPU/Memory usage.	moogfarmd may be processing an event or situation storm. Check moogfarmd health logging also for sign of message_queue build-up in any of the Moolets. Backlog should clear assuming storm subsides.
5	Has the memory of the moog_farmd java processed reached a plateau?	Moogfarmd may have reached its java heap limit. Check the -Xmx settings of moog_farmd. If not specified has moog_farmd reached approximately a

		quarter of the RAM on the server? Increase the -Xmx settings as appropriate and restart the moogfarmd service.
6	Is the Database tuned?	Check the `innodb-buffer-pool-size` and `innodb_buffer_pool_instances` settings in /etc/my.cnf as per Tuning section above. Ensure they are set appropriately and restart mysql if changes are made.
7	Check the server for any other high CPU or Memory processes or that which might be impacting the Database.	Something may be hogging CPU/Memory on the server and starving Farmd of resource. The events_analyser utility may be running or a sudden burst of UI or Graze activity may be putting pressure on the Database and affecting Farmd.
8	Run DBPool Diagnostics (see previous section) several times to assess current state of moogfarmd to database connections.	Moogfarmd database connections may be maxed out with long running connections - this may indicate a processing deadlock - perform a `kill -3 <pid>` on the moog_farmd java process to generate a thread dump (in moogfarmd.log) and send it to Cisco Support. Alternatively moogfarmd may be very busy with lots of short but frequent connections to the Database. Consider increasing the number DBPool connections for moogfarmd by increasing the top-level "threads" property in moog_farmd.conf and restarting the moogfarmd service.
9	Turn on MySQL slow query logging (see earlier section on how to do this)	Slow queries from a Moobot in moogfarmd may be causing problems and they should be reviewed for efficiency. Alternatively slow queries from other parts of the system may be causing problems (e.g. nasty UI filters). Slow queries may also be down to the sheer amount of data in the system. Consider enabling Database Split to move old data and/or using the Archiver to remove old data.
10	Check moogfarmd Situation resolution logging using: `grep "Resolve has been running for" /var/log/moogsoft/moogfarmd.log`	If this logging shows non-zero upward trend in "Resolve" time then Farmd is struggling with the number of "In Memory" Situations for its calculations. Check the moogfarmd health logging for the current count of in memory situations and consider reducing the retention_period setting in moog_farmd.log (will need a moogfarmd restart) and/or closing more old Situations.
11	Is moogfarmd memory constantly growing over time and a memory leak is suspected? Note that Farmd memory does typically increase for periods of time then is trimmed back via Java garbage collection and Sigaliser memory purge (via retention_period property).	Take periodic heap dumps from the moog_farmd java process and send them to Cisco support so they can analyse the growth. Use the following commands: `DUMPFILE=/tmp/farmd-heapdump-$(date`

		+%s).bin sudo -u moogsoft jmap -dump:format=b,file=$DUMPFILE $(ps -ef|grep java|grep moog_farmd|awk '{print $2}') bzip2 $DUMPFILE **Notes:** • jmap needs java jdk to be installed. "yum install jdk" should suffice to install this. • generating a heap dump is like to make the target process very busy for a period of time and also triggers a garbage collection so the memory usage of the process may well reduce. • heapdump files may be very large.

If the system is showing signs of slow UI performance, such as long login times, spinning summary counters, or other, then the problem is likely with Tomcat and/or the database. The following diagnostic steps will help you track down the cause:

Step	Description	Possible Cause and Resolution
1	Check catalina.out for any obvious errors or warning.	Cause may be evident from any warnings or errors.
2	Check browser console or any errors or timing out requests.	Possibly a bug or more likely that the query to the Database associated with the request is taking longer that 30secs (the default browser timeout). Root cause of this should be investigated.
3	Check network latency between browser client machine and server using ping.	Latency of =>100ms can make login noticeably slower.
4	Check the CPU/Memory usage of the server itself.	If the server, as a whole, is running close to CPU or Memory limit and no other issues can be found (e.g. rogue processes or memory leaks in the Cisco Crosswork Situation Manager components) then consider adding more resource to the server or distributing the Cisco Crosswork Situation Manager components.
5	Check MoogSvr/Moogpoller/Graze counter logging in catalina.out	Tomcat may be processing a high number of requests or bus updates. If Moogpoller count is zero then something may be wrong with Tomcat RabbitMQ connection. Check RabbitMQ admin UI for signs of message queue build-up.
6	Check whether Tomcat java process is showing constant high CPU/Memory usage.	Tomcat may be processing the updates from an event or situation storm. Backlog should clear assuming storm subsides.
7	Has the memory of the Tomcat java processed reached a plateau?	Tomcat may have reached its java heap limit. Check the -Xmx setting in /etc/init.d /apache-tomcat. Increase the -Xmx settings as appropriate and restart the apache-tomcat service.
8	Is the Database tuned?	Check the **innodb-buffer-pool-size** and **inn**

		odb_buffer_pool_instances settings in /etc /my.cnf as per Tuning section above. Ensure they are set appropriately and restart mysql if changes are made.
9	Check the server for any other high CPU or Memory processes or that which might be impacting the Database.	Something may be hogging CPU/Memory on the server and starving Tomcat of resource. The events_analyser utility may be running or a sudden burst of Farmd or Graze activity may be putting pressure on the Database and affecting the UI.
10	Run DBPool Diagnostics (see previous section) several times to assess current state of Tomcat Database connections.	Tomcat database connections may be maxed out with long running connections - this may indicate a processing deadlock - perform a `kill -3 <pid>` on the Tomcat java process to generate a thread dump (in catalina.out) and send it to Cisco Support. Alternatively Tomcat may be very busy with lots of short but frequent connections to the Database. A Graze request bombardment is another possibility (Graze does not currently have a separate DB Pool). Consider increasing the number DBPool connections for Tomcat by increasing the related properties in servlets.conf and restarting the apache-tomcat service.
11	Turn on MySQL slow query logging (see earlier section on how to do this)	Slow queries from nasty filters in the UI may be causing problems and they should be reviewed for efficiency. Alternatively slow queries from other parts of the system may be causing problems (e. g. inefficient Moobot code). Slow queries may also be down to the sheer amount of data in the system. Consider enabling **Database Split** to move old data and/or using the **Archiver** to remove old data.

12	Is Tomcat memory constantly growing over time and a memory leak is suspected?
	
	Note that Tomcat memory does typically increase for periods of time then is trimmed back via java garbage collection.
		Take periodic heap dumps from the Tomcat java process and send them to Cisco support so they can analyse the growth.
		Use the following commands:
		
		`DUMPFILE=/tmp /tomcat-heapdump-$(date +%s).bin`
		`sudo -u tomcat jmap -dump: format=b, file=$DUMPFILE $(ps -ef|grep java|grep tomcat|awk '{print $2}')`
		`bzip2 $DUMPFILE`
		
		Notes:

		• jmap needs Java JDK to be installed. "yum install jdk" should suffice to install this. • generating a heap dump is likely to make the target process very busy for a period of time and also triggers a garbage collection so the memory usage of the process may well reduce. • heapdump files may be very large

Control Cisco Crosswork Situation Manager Processes

This topic describes the commands for starting, stopping or restarting individual Cisco Crosswork Situation Manager processes.

Dependencies

Integrations (LAMs), moogfarmd, and Tomcat depend on the system processes: MySQL RabbitMQ, Nginx, and Elasticsearch. So when starting Cisco Crosswork Situation Manager processes:

1. Start or verify the following are started:
   - MySQL
   - RabbbitMQ
   - Nginx
   - Elasticsearch
2. Start or restart integrations (LAMs), moogfarmd, or Tomcat.

Similarly, if you plan to stop any one of MySQL, RabbitMQ, Nginx, or Elasticsearch, stop integrations (LAMs), moogfarmd, and Tomcat first.

Init Scripts for RPM Installs

If you you performed an RPM installation as root, use the service init script to start and stop Cisco Crosswork Situation Manager processes:

```
service <service-name> start|stop|restart
```

The service names are as follows:

- MySQL: `mysqld`
- RabbitMQ: `rabbitmq-server`
- Nginx: `nginx`
- Elasticsearch: `elasticsearch`
- Tomcat: `apache-tomcat`
- `moogfarmd`
- For LAMs , refer to the individual LAM references for the service names.

For more information, see the documentation on managing system services for your operating system.

Process Control for Non-root Installations

For customers who follow the Single Host Installation for Non-root Users procedure, Cisco Crosswork Situation Manager includes a process control utility to let you:

- Start a process
- Stop a process
- Restart a process
- Check the status, running or stopped, of a process.

The process control utility resides at `$MOOGSOFT_HOME/bin/utils/process_cntl`.

When you install Cisco Crosswork Situation Manager as a user other than root, you choose a user to run the installation and initialize the system. Use the same user credentials when controlling Cisco Crosswork Situation Manager components to ensure that you have the proper permissions and access .

process_cntl Command Line Reference

`process_cntl` uses the following syntax:

```
process_cntl [ [--process_name] <name>] [--loglevel] <loglevel>] [--
service_instance <instance>] {start|stop|status|restart|help}
```

The arguments for `process_cntl` are as follows:

Argument	Input	Description
`-h,--help`	-	Display the `process_cntl` syntax and option descriptions.
`--loglevel`	DEBUG | INFO | WARN	Log level controlling the amount of information that process control logs. Defaults to WARN. This flag only works for Cisco Crosswork Situation Manager processes.
`--process_name`	process name	The name of the process to control. You can specify one of the core processes: • `rabbitmq` - RabbitMQ message broker • `mysql` - MySQL database • `nginx` - Nginx web server. • `elasticsearch` - Elasticsearch search engine. • `apache-tomcat` - Apache Tomcat servlet container. • `moog_farmd` - Cisco event processing process. Alternatively specify an integration (LAM). Run `process_cntl -h` for a full list of integrations and syntax. See Implementor Guide for a brief description of the packages.
`--service_instance`	instance name	The name of the process instance if there is more than one running on the system.
`start`	-	Start a stopped process.
`stop`	-	Stop a running process.
`status`	-	Display the status of the process: running or stopped.
`restart`	-	Restart a running process