Weathervane User's Guide

Harold Rosenberg (hrosenbe@vmware.com)

This manual applies to Weathervane version 1.0.8

1 Introduction

1.1 Overview

Weathervane is an application-level performance benchmark designed to allow the investigation of performance tradeoffs in modern virtualized and cloud infrastructures. Weathervane can also be used as a tool for training or demonstration of software or hardware services. This document is the user's guide for Weathervane. It gives an overview of the design and implementation of Weathervane, and describes the tasks that must be performed to set-up and run the benchmark.

Weathervane consists of an application, a workload driver that can drive a realistic and repeatable load against the application, and a run-harness that automates the process of executing runs and collecting results and relevant performance data. It can be used to investigate the performance characteristics of cloud and virtual infrastructures by deploying the application on the environment of interest, driving a load against the application, and examining the resulting performance metrics. A common use-case would involve varying some component or characteristic of the infrastructure in order to compare the effect of the alternatives on application-level performance.

The Weathervane application is a web-application for hosting real-time auctions. This auction application uses a scalable architecture that allows deployments to be easily sized for a large range of user loads. A deployment of the application involves a wide variety of support services, such as caching, messaging, NoSQL data-store, and relational database tiers. Many of the services are optional, and some support multiple provider implementations. In addition, the number of instances of some of the services can be scaled elastically at run time in response to a preset schedule or to monitored performance metrics. The flexibility of the application deployment allows the investigation of a wide variety of complex infrastructure-related performance questions.

The Weathervane workload driver can support the complex workloads needed to drive this application, including support for asynchronous behaviors modeling the effect of embedded JavaScript and those requiring complex data-driven behaviors, as in web applications with significant inter-user interaction.

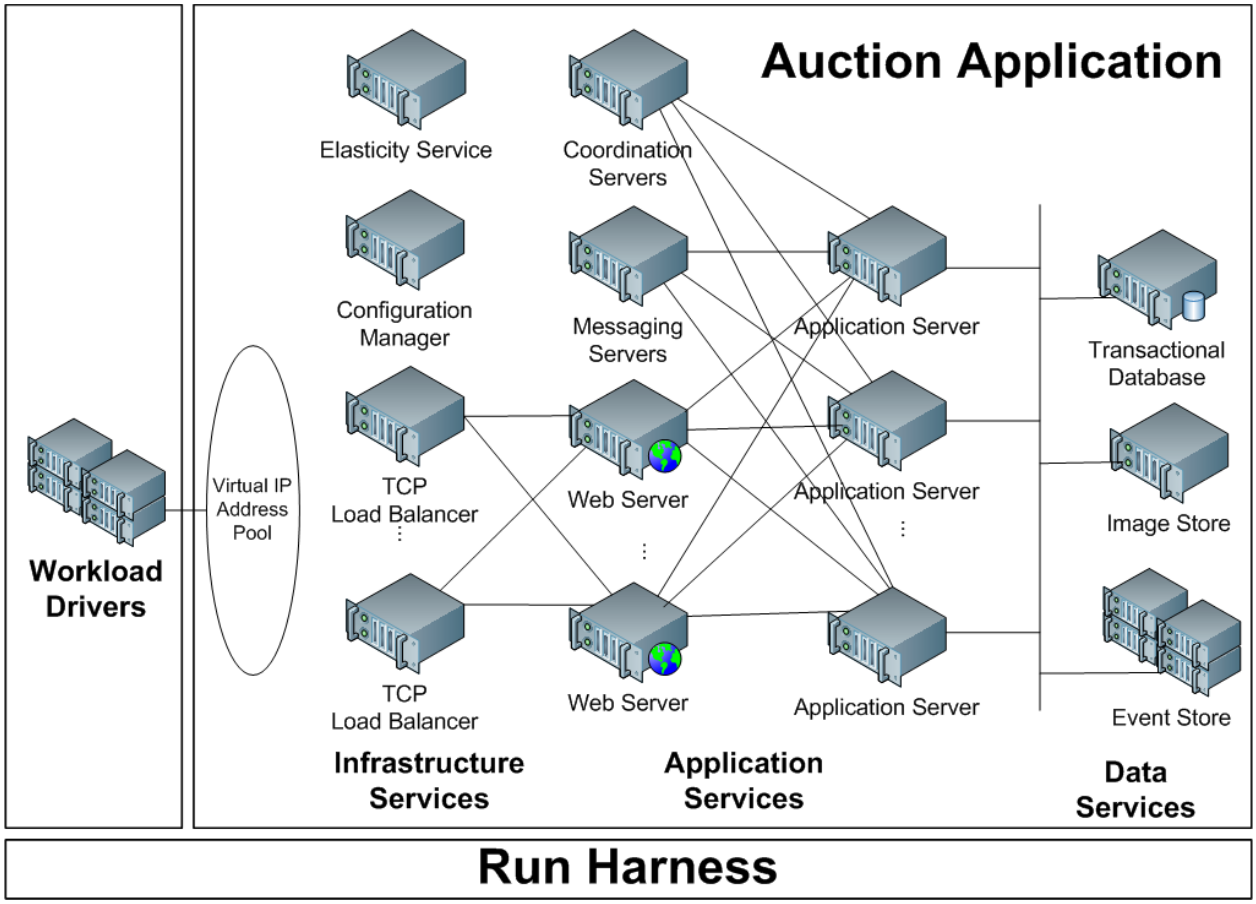
The Weathervane run harness manages the complexity of deploying and running a large multi-tier application benchmark. The harness takes as input a configuration file that describes the deployment configuration, the user load, and many service-specific tuning parameters. The harness is then able to power on

virtual machines, configure and start the various software services, deploy the software components of Auction, run the workload, and collect the results, as well as the log, configuration, and statistics files from all of the virtual machines and services. The harness also manages the tasks involved in loading and preparing the data in the data services before each run.

Figure 1 shows the logical layout of a full Weathervane deployment. The Auction application uses many software services and is deployed across multiple VMs or Docker containers. The Auction application is discussed in more detail in Section 5. The workload driver can be scaled out across multiple VMs in order to drive large loads against the application.

The remainder of this section discusses the primary use-cases for Weathervane and highlights some of its key features.

Figure 1 Full Weathervane Deployment



1.2 Uses of Weathervane

Weathervane as a performance benchmark

Weathervane has many of the features of a typical industry-standard application-level benchmark.

- It includes an application which is representative of a certain class of application that is common in production deployments.
- It defines a specific workload made up of operations performed by simulated users interacting with the application.
- It defines quality-of-service (QoS) and other measures that are used to determine whether a given run can be counted as an acceptable (passing) run.
- It collects and reports application-level performance metrics that can be used to compare multiple runs.

However, unlike most industry-standard benchmarks, Weathervane does not include strict run and reporting rules that control the configurations that can be used and the tuning that can be performed when running the benchmark. This gives users great flexibility in selecting the Weathervane configuration that is appropriate for their specific needs. It also means that Weathervane results collected by different users can almost never be compared against each other in any meaningful way. Note however it is possible to layer run and reporting rule on top of Weathervane in order to create comparable results.

Weathervane as a tool for testing and demonstration

1.3 Key Features

Simple Setup and Deployment

Support for Docker Containers

To be added

For more information on this topic see Section 10.

Support for Variable Loads

To be added.

For more information on this topic see Section 12.

Support for Application Elasticity

To be added.

For more information on this topic see Section 13.

2 What's New in Release 1.0.4

2.1 Overview

This section provides an overview of the changes that are visible to the user of Weathervane, or that might change performance results from the previous release. Information about past releases is contained in Section 16.

2.2 Key Changes

2.3 Known Issues

The following are known issues in this release:

- In some cases, changing deployment configurations will require reloading the database, even if the data services in the new configuration use the same data disks. For example, reloading may be needed when switching from running the data services directly on the OS to running them in Docker containers on the same hosts.
- Only PostgreSQL is supported as a database server when running the database tier in a Docker container.
- Only Nginx is supported as a web server when running the web tier in a Docker container.

3 Creating a Weathervane Host

3.1 Introduction

The process of setting up Weathervane starts with creating a Weathervane host. A Weathervane host is a virtual machine or server running Centos 7 which has been configured to run the Weathervane workload driver, run harness, and application components. It is possible to run all Weathervane components in a single Weathervane host. In a more typical deployment, the components will be spread across multiple hosts.

For the current release, only Centos 7 is supported for Weathervane hosts. When the application components are run as Docker containers, they may be run on any Docker host. However, if the Docker host is not configured as a Weathervane host it will be necessary to manually configure the mount points for the volumes used by the Auction data-services components.

Creating a Weathervane host is relatively straightforward:

- Create a Centos 7 VM
- Install any tools or drivers required by your virtual infrastructure

- Clone the Weathervane git repository, or unpack a Weathervane release tarball, in /root
- Build the Weathervane executables.
- Run the autosetup script provided with Weathervane.
- Reboot

Additional detail on each of these steps is provided in the following sections.

3.2 The Centos 7 Host

In order to create a Weathervane host, you must install Centos 7 in either a VM or on a bare-metal server. The Centos 7 installation may be a minimal install or a full desktop install. In fact, you may want to create a Weathervane host with a full desktop install for running the harness, and a second with a minimal install for cloning to VMs for running the various Weathervane services.

In order to be able to perform an initial run with the default settings, as described in Section 4, you should create the host on physical or virtual hardware with at least 2 CPUs, 8GB of memory, and at least 20GB of disk space. You will want to customize the hardware configuration when cloning the host for larger deployments of Weathervane. More details about this are discussed in Section 7.3.

The Weathervane host must be attached to the internet when running the auto-setup script. Once the host is configured it may be moved to a private network for running the benchmark.

After completing the OS installation, you should update all software by running the command *yum update* as the root user.

VMware vSphere Specific Considerations

When creating a VM for the Weathervane host, select Linux as the Guest OS Family, and Red Hat Enterprise Linux 7 (64-bit) as the Guest OS Version. This is necessary in order for proper operation of customization scripts when cloning the VM.

3.3 Install appropriate guest tools

Introduction

Many physical or virtual infrastructures will require the installation of drivers or guest tools for optimal performance. You should install these tools before installing Weathervane.

vSphere

On Centos 7, you can install the required VMware tools by executing the following command as the root user:

- *yum install -y open-vm-tools*

3.4 The Weathervane Repository

Weathervane can be acquired by cloning the git repository from GitHub.com, or by obtaining a release tarball from GitHub and unpacking in on the Weathervane host.

3.4.1 Cloning the Git Repository

NOTE: The location information will change once the Weathervane repository is configured on github.

The Weathervane repository can be obtained by cloning it from GitHub using git. Log into Centos 7 as root, and make sure that you are in the /root directory. Depending on the packages you chose when you installed the OS, you may have to install git using the command `yum install –y git`. Then clone the repository by issuing the following command:

git clone ssh://git@git.eng.vmware.com/private/hrosenbe/weathervane.git

This will create a clone of the repository in /root/weathervane.

3.4.2 Obtaining and Unpacking a Release Tarball

Weathervane can also be obtained as a release tarball from the Release area of the GitHub repository. A release tarball is a snapshot of the repository at known good point in time. Releases are typically more heavily tested than the latest check-in on the master branch. Note that the release tarballs are not binary releases. You still need to build Weathervane as discussed in the next section.

To install Weathervane from the tarball, perform the following steps:

1. Log into your Weathervane host as the root user.
2. Place the Weathervane tarball in the directory /root.
3. From within the root directory, unpack the tarball with the following command: *tar zxf weathervane-x.y.z.tar.gz*, where x.y.z is the release number in the name of the tarball.

This will create a new directory at /root/weathervane and will unpack the tarball into that directory.

3.5 Building the Weathervane Executables

In order to build the Weathervane executables, change directory into the /root/weathervane directory created in Section 3.4, and then issue the following command:

./gradlew clean release

Depending on the packages that you chose when installing Centos 7, you may get an error that no java command could be found in your path. If so, install java using the command:

yum install –y java-1.8.0-openjdk*

and then try again.

The first time you build Weathervane, this will download a large number of dependencies. Wait until the build completes before proceeding to the next step.

3.6 Running the auto-setup script

After cloning the Weathervane repository or unpacking the tarball, the next step is to run the auto-setup script to configure the host to run all of the Weathervane components. As mentioned previously, the host must be connected to the internet in order for this process to succeed.

In order to run the auto-setup script, perform the following steps:

1. Log into your Weathervane host as root.
2. Install Perl, which is needed for running the autosetup script. The command to install Perl is: *yum install –y perl*

   Note: Depending on which packages you chose when installing Centos 7, Perl may already be installed on your server.
3. Change directory to /root/weathervane.
4. Run the script using the command: *./autoSetup.pl*
5. Reboot the VM before proceeding to the First Run described in Section 4.

The auto-setup script may take an hour or longer to run depending on the speed of your internet connection and the capabilities of your host's hardware.

4 First Run

4.1 Introduction

This User's Guide provides a great deal of information about deploying and running the Weathervane benchmark. The large scope of the benchmark and the number of configuration choices for its deployment can be confusing for first-time users. It will be helpful to become familiar with Weathervane by running the benchmark in a contained setting before embarking on a complete deployment. This section discusses how to perform some initial runs of Weathervane in a single Weathervane host.

4.2 Weathervane Execution Overview

The Weathervane benchmark is run from the host that is acting as the primary workload-driver. Weathervane comes with a run harness that automates the process of running the benchmark. Each run, or series of runs, is started by executing the run harness on the primary Weathervane host. Typically, you will also run the workload driver and DNS server on this host.

The Weathervane run harness automates the execution of the benchmark. It will set up the services, prepare the database, and run the workload. At the end of a run it can automatically collect performance statistics from all VMs and hosts,

and parse the data to produce a summary of each run. It can also automate the execution of multi-run experiments.

One thing to be aware of when running Weathervane is that a run of the benchmark can take anywhere from 20 minutes to an hour. The process of preparing the database and waiting for the start of the first auctions in the Auction application may take up to 5 minutes per run. The benchmark itself will require a runtime of at least 25 minutes to get a steady state of reasonable duration, although it may also be set to run for arbitrarily long periods if desired.

4.3 First Run

The process of setting up Weathervane is relatively straightforward. Nevertheless, it will be useful to see Weathervane and the Auction application in action before going through the full setup. This section explains the steps involved in performing a first run of the benchmark. This first run also provides an opportunity to explore the Auction application.

To run Weathervane:

1. Use a Weathervane host created as described in Section 3 or a clone of that host.
2. Log into the host's console as root.
3. Open a terminal and change directory into the Weathervane directory: `cd Weathervane`
4. Start the DNS server: `systemctl start named`
5. Run the benchmark: `./weathervane.pl`

weathervane.pl is the interface to the Weathervane run-harness. The run harness manages starting the software components, running the workload driver, collecting logs and performance data, and cleaning up after each run. The Weathervane run harness gets its parameters from a configuration file. The default name for this configuration file is *weathervane.config*, located in the */root/weathervane* directory. Weathervane comes with a fully annotated weathervane.config file which contains explanations of the most important parameters for the run-script. The initial run uses the default values for all of the parameters. This drives a load of 300 users in a 26-minute run (10-minute ramp-up, 15-minute steady-state, 1-minute ramp-down). It will use PostgreSQL as the database, and MongoDB as the image-store.

When the run-script executes, it will print messages about the progress of the set-up and execution of the benchmark. You will see the run-script do the following:

- Do some cleanup to make sure that nothing is left over from previous runs and all components are cleanly shut down.

- Configure all software services for the current run. This includes editing configuration files to set tuning parameters specified in the configuration file.
- Check whether the data services are loaded with the proper data for the run. The data services are the relational database (PostgreSQL in this run), the NoSQL data-store (MongoDB), and the image-store (also MongoDB for this initial run).
- Load the necessary data into the data services.
- Prepare the data for the run. This involves configuring the auctions that will be active during the run.
- Start running the workload. The workload will run for 26 minutes.
    - At the default log level, no performance data is collected. When performance data-collection is enabled the run-script will start data-collection once the steady-state starts. Once the steady-state ends, the run-script will stop data-collection.
- When the workload driver finishes running, the run harness will parse the results, determine whether the run passed the QOS requirements (described later), and create a summary of the run in a csv file.

During the run, you should open a connection to the application in a web browser, either locally at https://www.weathervane or remotely, replacing www.weathervane with the IP address assigned to the VM. Sign into the application with the username 'guest@foobar.xyz' and the password 'guest'. You will then be logged into the application. Once logged in, you can join an active auction and explore the other functionality of the Auction application. Join some active auctions, and watch the bidding by the simulated users. You can even bid on items yourself. Note that some of the links on the pages are placeholders for future functionality, and some functionality of the application is not reflected in the interface.

After the run, you should explore the various output files created during the run. These include:

- The file *weathervaneResults.csv* which is created in /root/weathervane. For every run of the benchmark, the run-script places one line in this file with a summary of the run configuration, results, and collected performance statistics. You can install LibreOffice on your run-harness VM to view the file locally, or you can copy it to your local workstation.
- The directory /root/weathervane/output/*n*, where *n* is the run number. For the first run, the run number will be 0. This directory has all of the raw data collected during the run, including the workload-driver output; the various application logs; and the statistics output-files. Some files of particular interest are:

- output/0/console.log. This file will contain the information that was printed to the terminal during the run, including the run results.
- output/0/run.log. This file will contain the workload-driver output for the run.
- output/0/setupLogs. This directory contains logs that the run harness creates when setting up for a run. These logs may be useful if a run fails.
- output/0/cleanupLogs. This directory contains logs that the run harness creates when cleaning up after a run. These logs may be useful if a run fails.
- output/0/configuration. This directory contains the configuration information for the workload driver and a copy of the Weathervane configuration parameters. If the log level is greater than 0, it will also contain configuration information for all of the services used during the run.
- output/0/statistics: This directory will contain the performance statistics gathered when a run is performed with the log level greater than 1.

You can try additional runs by editing the parameters in Weathervane.config. You can also override the configuration-file parameters using parameters on the command-line. For example, the command '*./runWeathervane.pl –users=60'* will run the benchmark using all of the parameters from the configuration file but with 60 simulated users. Note that you will not be able to run more than 300 users with the default data loaded on the Weathervane VM. You can get an overview of the command-line parameters by running `./runWeathervane.pl –help`.

Once you are comfortable with the benchmark, you can proceed to the full set-up as discussed in the following chapters. You should start the full set-up with a fresh Weathervane host in order to avoid carrying over any changes you may have made during your initial explorations.

5 Weathervane Auction Application

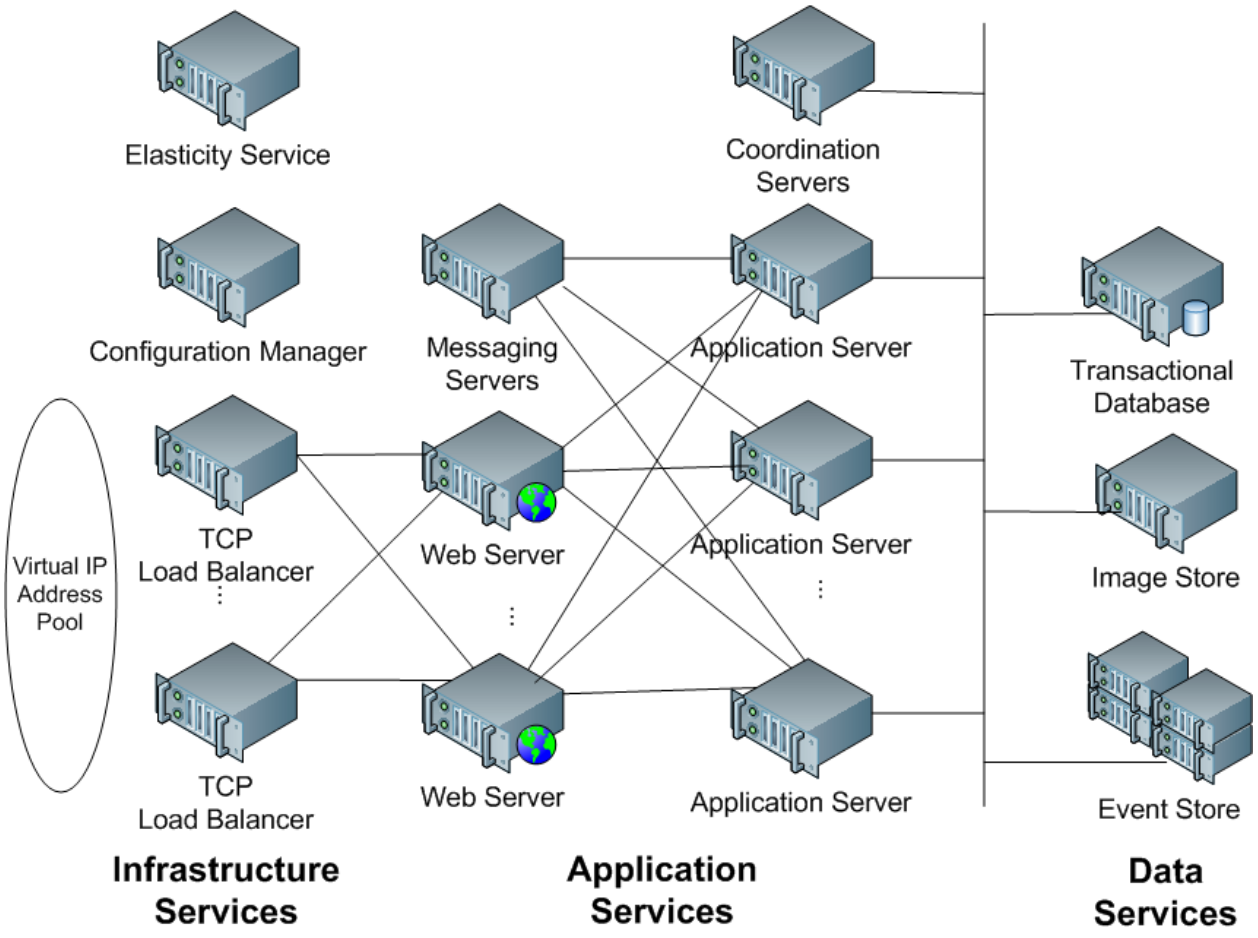
5.1 Overview

In order to successfully create a full Weathervane deployment, it is necessary to have a more complete understanding of the Auction application. The Auction application is designed in such a way that it can be deployed in many different configurations. The configurations may be as small as one VM hosting all of the services that make up the application, while a large configuration may involve dozens of VMs.

Figure 2 shows a complete deployment of the Auction application. In addition to the application servers which run the application logic, a deployment of Auction

includes a number of support services and data sources. A description of these services and their role in the Auction application is given in Section 5.2.

Figure 2 Auction Application Deployment



Weathervane provides a number of options when mapping the application services onto the hosts on which they run. It is possible to run multiple services on a single Weathervane VM, bare-metal host, or Docker host. When using Docker containers it is possible to run multiple instances of the same service on a single Docker host. The Weathervane run-harness manages the configuration of services to avoid any issues with TCP port-number conflicts. The mapping of services onto hosts is discussed in Section 7.4.

Each service type has a number of configuration and tuning options that are controlled by parameters in the Weathervane configuration file. These parameters are discussed in Section 8.7.

5.2 Auction Application Services

A deployment of the Auction application may consist of up to ten separate service tiers. Not all of these tiers are required in any given deployment of Weathervane. The run harness will adjust the configuration of the services to

enable proper operation with different service combinations. This section gives a brief overview of the services that can be used in a Weathervane deployment.

The services used in an Auction deployment can be divided into three categories: infrastructure services, application services, and data services. Infrastructure services are the services that play a role in handling and routing the client requests, but do not operate on those requests. Services involved in managing the topology of the deployment are also classified as infrastructure services. Application services are those services that either implement application logic or serve application content to the clients. Data services are the services that store data for the application, but do not directly handle client requests. The data services are accessed only by the application services, not directly by the clients. The services in each category and their roles in a Weathervane deployment are as follows.

- Infrastructure Services
    - Virtual IP Management Service (optional): The Weathervane workload-driver drives an equivalent user load to each of the client-facing front-end services. In Figure 2 the front-end service is the load-balancer, but in other deployments the front-end service may be the web servers or an application server. By default, the run-harness configures the workload driver to drive load to the actual IP addresses of the hosts containing the front-end services. While this works, it does not represent a production-grade solution. If any of the front-end nodes go down, then then application will be unavailable to users using that IP address. Weathervane supports an optional virtual-IP mode, in which a set of virtual IP addresses is configured for the front-end services. The assignment of these IP address to the service hosts is managed by a Virtual Router Redundancy Protocol (VRRP) service. Weathervane uses keepalived as the interface to this service. If one of the front-end nodes goes down, keepalived will automatically reassign its virtual IP address to one of the other nodes. When using the virtual-IP mode, the keepalived service always runs on each of the hosts containing one of the front-end services. Note that virtual-IP mode is more complex to configure correctly, particularly when testing multiple configurations. As a result, it is best to reserve its use for situations where you are interested in performance in the presence of node failures.
    - Load Balancer (optional): The Load Balancer acts as a front-end node to balance client requests among multiple web servers. If a deployment without web servers is used, the load-balancer can be used to balance client requests among multiple application servers. The load-balancer implementation used by Weathervane is HAProxy.

- Configuration Manager (optional): The configuration manager is a microservice whose function is to track the current configuration of the application deployment and manage configuration changes when application elasticity is used. It has a REST API that can be used to request the addition or removal of services from the configuration. When configuration changes are requested, the configuration manager handles configuring and starting new services, as well as integrating the services into the relevant load-balancing domains. When new application servers are added, the configuration manager will pre-warm the JVM (class loading and compiling), as well as relevant application-level caches. This prevents poor performance when load is initially directed to the new application server. The configuration manager uses very little resources, and so can be run on the same node as other services. The configuration manager is required when using application elasticity (see Section 13.2), or when pre-warming of all app servers is requested at the start of a run (see the description of the "prewarmAppServers" parameter in Section 0). Pre-warming the app servers allows runs to complete successfully with a minimal rampUp period.
- Elasticity Service: The elasticity service is a microservice whose role is to determine when changes in the application's configuration are necessary, and to request that the configuration manager implement those changes. Conceptually, the elasticity service could monitor application performance metrics, decide where services should be added or removed based on the metrics, and interact with both the infrastructure APIs and the configuration manager to add resources and integrate the required services. The current implementation of the elasticity service uses a fixed schedule of configuration changes, rather than monitoring performance metrics, and does not interface with any infrastructure APIs. For more detail see Section 13.2.

- Application Services
  - Web Server (optional): The web server plays a number of roles in a Weathervane deployment. It serves static content such as html and JavaScript scripts in response to requests from clients, it serves and caches images, it terminates SSL sessions with clients, and it load-balances requests among multiple application servers. The web server is optional, as all of these tasks except load-balancing can be handled by the application servers. However, the web server does provide significant offload when using multiple application servers. Weathervane supports the use of Apache httpd 2.4 and Nginx web servers. Only the free version of Nginx is supported.

- Application Server. The application servers host the Java web application that is the heart of the Auction application. This application implements all of the business logic of the Auction application. The application is stateless, allowing it to be easily scaled-out horizontally over multiple VMs. Weathervane currently supports Apache Tomcat 8.5 as the application server.
- Message server. The messaging service provides the backbone for communication among the application servers. It is used to communicate key state changes, as well as to distribute work to various microservices. RabbitMQ is the only message server supported by Weathervane.
- Coordination Server: The coordination server is used by the auction management service when choosing a leader to handle the assignment of auctions to nodes, as well as to enable agreement on the assignment of active auctions to nodes. The coordination server runs the Zookeeper coordination service. In the Auction application the demand on Zookeeper will be small and therefore its memory footprint and CPU usage will be small as well. You can to safely run the coordination server instances together with other services, such as the database servers. There must be either 1 or 3 of these services.
  - If you are using elasticity features, you should not run the coordination services on the web or application server VMs. The current implementation of the Weathervane Elasticity Service does not power VMs on and off when services are added or removed, but future implementations may. This would cause the loss of other services running on those VMs.

- Data Services
  - Relational database server. The relational database server provides transactional data storage for the Auction application. It is used to store all data used by the Auction application that may be used within an atomic transaction. The relational databases supported by the current release are PostgreSQL and MySQL.
  - NoSQL document data-store. The NoSQL data store is used to store high-volume non-transactional data related to the operation of the Auction application. This includes historical data regarding bid and auction attendance, as well as image meta-data. In this role the NoSQL server is said to be acting as the event store. The NoSQL data-store can also be used to store the images used by the Auction application. When it is storing images, the NoSQL data-store is said to be acting as the image-store. MongoDB is the only NoSQL document data-store supported in the current release. MongoDB may be sharded or replicated across multiple VMs, but

sharded MongoDB replica sets are not yet supported by the Weathervane run-harness.

- File Server (optional): The file server is used to store images for the Auction application. When it is used, the file server, rather than the NoSQL data-store, is said to be acting as the image-store. Weathervane currently supports NFS as the file server.

6 The Weathervane Workload

6.1 Overview

In order to drive a load against the Auction application, Weathervane has a sophisticated workload-driver application that simulates the requests generated by users interacting with the application. The Weathervane run-harness configures the workload driver based on the configuration file, and starts and stops it during the run. Weathervane also includes support applications used for setting up the application data before running the workload.

6.2 Workload Services

A Weathervane deployment includes services are used to configure or drive load against the Auction application instances. These are:

- Workload Drivers: The Weathervane workload driver has been developed to drive HTTP-based loads for modern scalable web applications. It can simulate workloads for applications that incorporate asynchronous behaviors using embedded JavaScript, and those requiring complex data-driven behaviors, as in web applications with significant inter-user interaction. It uses an asynchronous design with a small number of threads supporting a large number of simulated users. Simulated users may have multiple active asynchronous activities which share state information, and complex workload patterns can be specified with control-flow decisions made based on retrieved state and operation history. These features allow us to efficiently simulate workloads that would be presented to web applications by rich web clients using asynchronous JavaScript operations. The workload driver creates a load of simulated users interacting with the Auction application during the run phase of a Weathervane run. It also collects the performance metrics operation-level mix, response-time and throughput metrics that are used when deciding whether a run has passed or failed. The workload driver can run on a single node or can be distributed to run on multiple VMs. In the latter case the primary workload-driver will collect the run statistics from all nodes. There must be at least one workload driver per workload.
- Data Managers: The data manager is responsible for loading and preparing the data in the data services of the Auction application. There is one data manager for each instance of the Auction application. The data

manager is used by the run harness during the data preparation phase of a benchmark run.

6.3 Simulated Users

The level of load driven against the Auction application by the workload driver is stated in terms of the number of active simulated-users. A simulated user performs a sequence of operations against the Auction application, modeling the behavior of a real user of the application. The sequence of operations is not fixed, but varies based on conditions at a number of decision points. The next operation performed at each decision point may depend on the user's operation history, the value of state information retrieved from the application by a previous operation, and on random chance. For the Auction application, the next operation may depend on the actions of other users, as reflected in the state of bidding on individual items. While a user's operation sequence at any given point in time will be random, the workload and application have been designed to ensure that the long-term percentage of each operation in the overall mix is constant. This helps ensure that the load profile driven by a simulated user is consistent from run to run. If overloading of one portion of the infrastructure causes the average operation mix to vary from the target percentages, then the run will be considered to have failed.

In addition to ensuring that the operation mix remains consistent, the workload driver has been designed to ensure that the average arrival-rate of operations per user remains constant from run to run. The combination of a consistent operation-mix and a fixed load per user ensures that load placed on an Auction deployment, and thus the underlying infrastructure, by a given number of users is consistent from run-to-run. This in turn ensures that performance metrics captured from runs at the same number of users are truly comparable.

6.4 Workload Pass/Fail

Weathervane uses a number of metrics to determine whether a run of the benchmark can be considered a valid, or passing, run. These are:

- Operation-Mix Percentages
- Response-time quality-of-service (QoS) requirements.
- Sanity Checks

As discussed in the previous section, each operation type performed by a simulated user must make up a fixed percentage of the overall operation mix, although the run-harness actually allows a small amount of deviation from the exact values. If the operation mix percentages for a run do not fall within the allowed tolerance, the run is considered to be failed.

Each operation type has a defined response-time limit. The Weathervane workload-driver computes the response time for each operation, and tracks the number of operations that fail to complete within the response-time limits. The default QoS requirement for Weathervane is that 99% of all operations in each

operation type must complete within the response-time limit. If more than 1% of any operation type fail to complete within the limit then the run is considered to have failed.

The Weathervane sanity checks cover non-performance related issues in the deployment. Currently the only sanity check is a test to make sure that the data disks for the MongoDB instances are not full. If any sanity check fails then the run is considered to have failed.

Failing the run requirements has two key implications:

- The user experience during the run was poor, and so the deployment and underlying infrastructure must be considered inadequate for the given load. The maximum supported load for a given deployment will be the highest load that passes all of the requirements.
- The collected performance metrics from a failed run should not be used to compare against other runs at an equivalent number of users. Some component may be causing the load placed on the deployment to deviate from the expected load for the given number of users.

6.5 Workload Operations

The Weathervane workload driver monitors quality-of-service (QoS) metrics for both the Auction application and the overall workload. The application-level QoS requirements are based on the 99th percentile response-times for the individual operations. An operation represents a single action performed by a user or embedded script, and may consist of multiple HTTP exchanges. The workload-level QoS requirements define the required mix of operations that must be performed by the users during the workload's steady state. This mix must be consistent from run to run in order for the results to be comparable. In order for a run of the benchmark to pass, all QoS requirements must be satisfied.

The QoS requirements for operations that make up the workload for the Auction application are shown in Table 1.

Table 1 Auction QoS Requirements

Operation	Required 99th Percentile Response-Time (seconds)	Percentage of Operation Mix
HomePage	2	0.92%
Login	3	0.91%
GetActiveAuctions	2	14.98%
GetAuctionDetail	2	7.53%
GetUserProfile	2	0.91%

UpdateUserProfile	2	0.46%
JoinAuction	3	6.87%
GetCurrentItem	3	7.09%
GetNextBid	36	39.63%
PlaceBid	2	2.77%
LeaveAuction	2	5.16%
GetBidHistory	2	0.46%
GetAttendanceHistory	2	0.46%
GetPurchaseHistory	2	0.92%
GetItemDetail	2	7.53%
GetImageForItem	5	0.40%
AddItem	2	1.37%
AddImageForItem	5	0.72%
Logout	3	0.91%

The GetNextBid operation is an asynchronous operation which updates a user's display in response to new bids from other users. As a result, its response-time requirements are dependent on characteristics of the application, such as the waiting period before items are deemed to be sold, and not end-user response-time.

7 Weathervane Deployment

7.1 Introduction

Planning and creating a complete deployment of Weathervane requires first answering the following questions:

- Which services will be used in the Auction application deployment?
- How many instances of each service type will be used, including the number of workload-driver nodes?
- Which service implementations will be used for each service type?
- What tunings and configuration options will be used for each service?

- What will be the mapping of services to bare-metal hosts, VMs, or Docker hosts?

Weathervane has defaults that simplify many of these questions. For example, other than adjusting JVM heap sizes, it is seldom necessary to change the tuning of the services or the default service implementations. Others, such as the number of services instance to use, will depend on the goals of the performance evaluation.

This section discusses the process of planning for and creating a full Weathervane deployment. Discussion of advanced deployment topics, such as creating deployments consisting of multiple independent instances of the Auction application, using Docker containers, or using application elasticity, is deferred to later sections. The actual specification of deployments using the Weathervane configuration file is discussed in Section 8.

7.2 Set-up Checklist

While many of the Weathervane set-up steps can be performed in any order, much of the set-up will be simplified if the configuration is performed in the following order. Each of these steps is discussed in more detail in later sections of this chapter.

1. Create a Weathervane host as described in Section 3. If you have already used a host for some initial runs as described in Chapter 4, you should start again with a fresh host to avoid carrying over any changes you may have made in your initial explorations.
2. Plan the Weathervane deployment. This includes selecting the number of hosts to be used, selecting the number of CPUs and amount of memory for each host, and the assignment of services to hosts.
3. Plan the mapping of services to hosts and configure the DNS Server.
4. Clone the Weathervane VM to create the VMs needed for the deployment. If you are using VMware vSphere, then this step includes creating a vCenter customization specification for the Weathervane VMs and then cloning the VMs using the customization specification and the appropriate IP addresses. If you are using a different virtual infrastructure (VI), you will need to follow the correct procedures for that VI to create necessary VMs and assign them the correct IP addresses and other network information. If you are running Weathervane on bare-metal, you will need to create Weathervane hosts individually by running the autoSetup script.
5. Configure password-less ssh to all hosts. If you are deploying all services on Weathervane hosts, then this will be done automatically. If you are using Docker hosts that are not Weathervane hosts, and you want to be able to collect host-level performance data from that host, then you will need to do this manually. If you wish to collect performance data from your VI hosts, such as esxtop data from VMware vSphere ESXi hosts, then you will need to set up password-less ssh to those hosts as well.

6. Ensure that the clocks on the VMs are synchronized using ntp or another time-synchronization mechanism. There are ntp-related parameters that instruct the run harness to manage this automatically. See Section 8.7.1.1 for more information.
7. Configure additional disks for the database, MongoDB, and, if using, the image-store file system.
8. Run the benchmark

7.3 Deployment Configuration Planning

Overview

A Weathervane deployment includes a number of different software components and a number of possible configurations for those components. The process of deployment planning involves selecting a deployment configuration; deciding how many services of each type to use; selecting how many hosts to use, where a host may be a VM, a Docker host, or a bare-metal host; selecting which software components to run on each host; choosing IP addresses for the hosts; and determining the CPU counts and memory sizes for the host. In this section we provide guidance on each of these tasks.

Deployment Configuration and Host Count

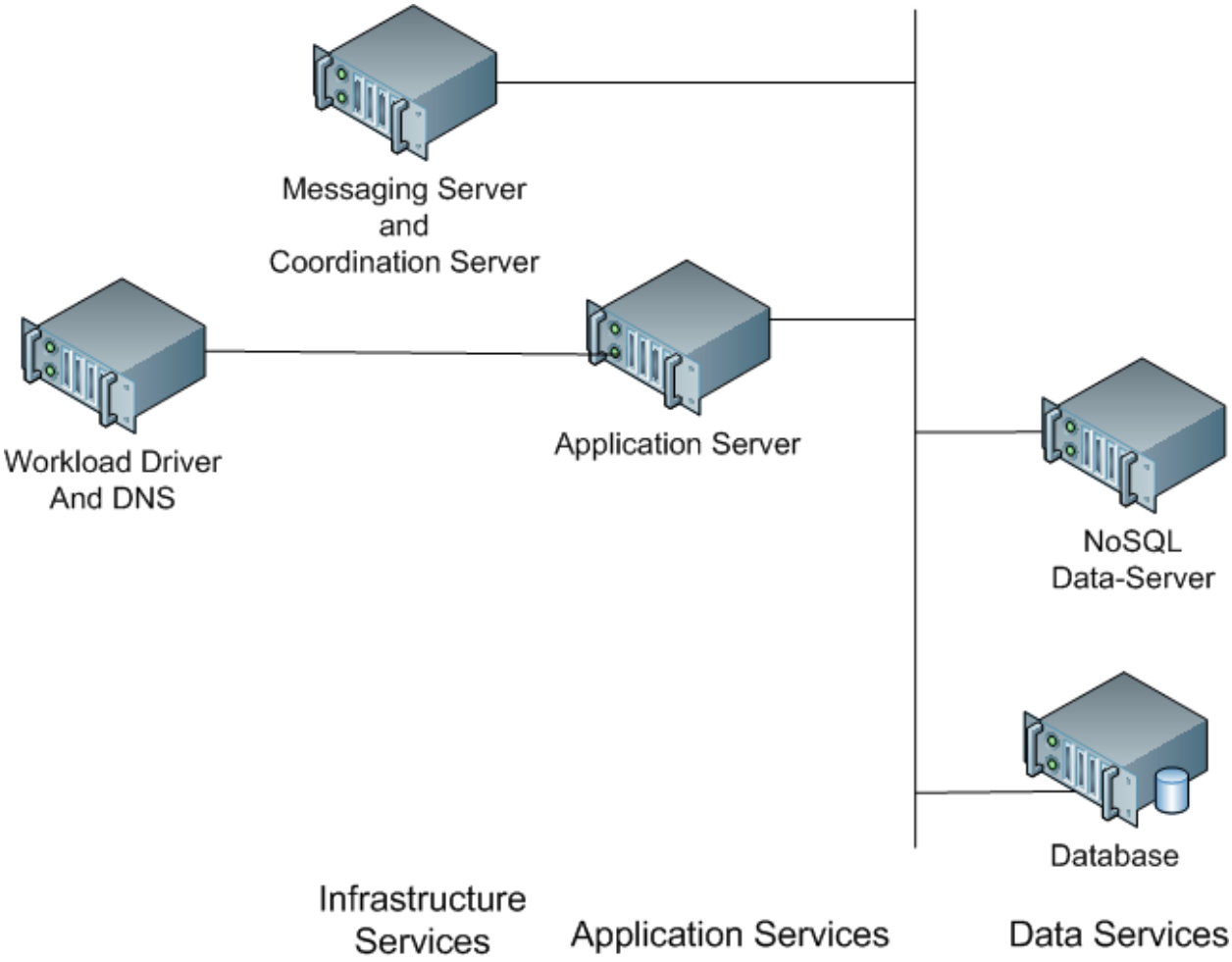
Weathervane provides significant flexibility when selecting a deployment configuration of the Auction application. Different deployment configurations will provide different performance capabilities. However, even if your goal is to scale to as large a load as possible on your infrastructure, it is not necessary to start with a complete scale-out deployment using all of the load-balancer, web-server, and application-server tiers.

Our recommendation is to start with a simple deployment and add additional VMs as needed to handle additional load or satisfy particular testing goals. The Weathervane run harness will perform the necessary reconfiguration when you change the deployment configuration. For example, if you start with an application server as the front end and later add a web server or load balancer with additional application servers, the run harness will change the application server configuration so that it can properly handle proxied requests from the webserver or load balancer. The same holds for switching from using MongoDB as the image-store to serving the images from NFS.

In initial testing you may want to run multiple software components on a single host. However, unless you plan to run only at small loads, a good starting configuration will be separate hosts for a workload driver, application server, database server, NoSQL server, and a single host for the coordination server and message server, for a total of six services on five VMs. A good next step would be to add a web server. The DNS Server is typically run on the the VM from which you run the run-harness. Additional tiers and VMs can then be added as dictated by testing needs or performance concerns. Figure 3 shows a logical

view of this configuration. In an actual deployment, the workload driver can be deployed on the same network as the other VMs.

Figure 3 Minimal Multi-VM Weathervane Deployment



Sizing the VMs

In order to assign the correct amount of CPU, memory, disk, and network capability to the hosts for each of the services involved in a Weathervane deployment, it will be useful to have some general guidelines for the resource demands for each service.

Table 1 shows the per-user CPU demands for the various software components of a Weathervane deployment. These demands were measured when running Weathervane on an ESXi host where the server had Intel Xeon E5-2687W CPUs (3.1Ghz). The per-user CPU utilizations will vary depending on the CPU type and other factors such as the deployment configuration, the OS and application-level tunings, storage performance, etc. However, this table can be used to determine an initial sizing. These demands represent the percentage of a single CPU used for each simulated user driven by the workload generator. Table 1 also shows the CPU demands normalized to that of the application server.

Table 2 CPU Demands

	CPU Used Per-User	Normalized CPU UT
Workload Driver	0.00033	113%
Application Server	0.00029	100%
Load Balancer	0.00007	22%
Web Server	0.00013	45%
Message Server	0.00002	7%
MongoDB with images	0.00004	15%
Database	0.00004	15%
MongoDB, images in filesystem		%
Coordination Server	2.8E-7	0%
Configuration Manager	1.1E-7	0%
Filesystem		

The figures in **Table 2** might be used as follows. On the application server, a single CPU will be able to support approximately 1/0.00029 = 3409 users. As a result, in a deployment with a single 1 vCPU application server VM, we would need at least 3409 * 0.00004 = 0.15 CPUs on the database VM. Another way to look at this is that for each database CPU, we can support about 7 application-server CPUs. In an actual deployment these ratios will vary depending on tuning, will change depending on load, and the saturation points of the components will actually occur at less than 100% utilization.

Table 2 shows an estimate of the per-user storage demands for the Weathervane components. These are provided for guidance when sizing a deployment. The actual storage I/O rates in a deployment will depend on tuning and memory sizes.

Table 3 Per-User Storage Demands

	Read TPS	Write TPS	MB Read/s	MB write/s
Application Server	0.000	0.000	0.000	0.000
Workload Driver	0.000	0.000	0.000	0.000
Load Balancer	0.000	0.000	0.000	0.000
Web Server	0.000	0.017	0.000	0.0004
Message Server	0.000	0.000	0.000	0.000
MongoDB with images	0.0004	0.023	0.00006	0.0014
Database	0.000	0.020	0.000	0.0002
MongoDB, images in filesystem				

Filesystem				

Table 3 shows an estimate of the per-user network demands for the Weathervane components. These are provided for guidance when sizing a deployment. These will vary depending on which tier is the front tier. These values assume a full deployment.

Table 4 Per-User Network Demands

	RX Packet/s	TX Packets/s	RX Mbps	TX Mbps
Application Server	2.0	1.7	0.009	0.011
Workload Driver	4.7	2.0	0.045	0.008
Load Balancer	6.9	3.5	0.053	0.051
Web Server	3.6	2.4	0.014	0.049
Message Server	0.13	0.12	0.0002	0.001
MongoDB with images	0.46	0.12	0.004	0.002
Database	0.38	0.36	0.0005	0.0005

Memory usage cannot be stated on a per-user basis. There are base memory requirements for the OS and the applications that are in some cases larger than the additional memory needed for the application load. There are also performance tradeoffs between memory size and I/O rates. Table 4 gives general recommendations on memory sizes for the various components, but experimentation will be needed on your testbed to determine appropriate sizing.

Table 5 Memory Usage

	Memory Size recommendations
App Server	3GB of Java Heap per vCPU, plus 512MB for OS. Remember to set the AppServerJvmOpts parameter.
RabbitMQ	2GB total will be probably be sufficient for most numbers of users
Database	Will vary with benchmark scale. As the size of the relational database memory is small, a good starting point is 8GB.
Workload Driver	2GB heap per vCPU should be sufficient. Remember to set the DriverJvmOpts parameter.
MongoDB	Will vary with benchmark scale. Setting to 32GB is a good start.

7.4 Mapping Services to Hosts

Overview

The various services used in a Weathervane deployment must all run on some type of host. A host may be a VM, a bare-metal host, or a Docker host. In order for the run harness to manage the run, it must know on which host each service should run. This is achieved by a two-step process in which services are mapped to host names, and then the host names are mapped to IP addresses.

One common approach is to map all services to different host names, and then control the assignment of services to hosts through the IP address mapping. The IP-address mapping is typically managed by configuring a DNS server to be used by the hosts in a deployment. Multiple services can be mapped to the same host by assigning the same IP address to their associated host names. The process for setting up a Weathervane host, discussed in Section 3, will install a DNS server and give it a default configuration. It is also possible to use the /etc/hosts files on the hosts to manage the IP mapping, but the Weathervane run harness does not provide support for this method.

The process of using the DNS server to manage the assignment of services to hosts is necessary when using the convention-based configuration method discussed in Section **Error! Reference source not found.**. Convention-based configuration assigns a host name to each service based on conventions discussed in that section. The mapping of services to hosts then depends on the configuration of the DNS server.

If it is not possible to use the DNS server, or if finer control over the configuration is desired, the host name for each service can be specified explicitly using the explicit configuration syntax discussed in Section 8.6.

Configuring the DNS Server

The Weathervane deployment depicted in Figure 1 shows most of the services used during a Weathervane run. Not depicted there are the data manager and DNS services. The services can be divided into those required to configure and run the workload, and those specifically associated with the Auction application.

The DNS server can run on only one host. Typically, the host on which you run the run harness also hosts the DNS server. Whichever host is selected, you should ensure that the DNS server always starts on reboot with the command `systemctl enable named`. The DNS server is disabled on Weathervane hosts by default.

The DNS server is used to associate the hostnames used by the Weathervane run-harness and application with the IP addresses of those hosts. The Weathervane host comes with the DNS server partially configured. However, the configuration of the DNS server must be customized for your Weathervane deployment.

The DNS server on the Weathervane host comes with the following set-up:

- The DNS server does not start automatically when the VM boots up.
- The DNS server forwards hostnames that it cannot resolve to a Google Public DNS server at IP address 8.8.8.8. Any hostname outside of the Weathervane domain will be forward to this server.
- The records for all of the service hostnames (i.e. AuctionDriver1, AuctionDB1, etc.) resolve to a localhost IP address, which is 127.0.0.1.

This configuration must be customized for your Weathervane deployment. This is done on the VM on which you will run the DNS server by editing the DNS configuration files.

1. Log into the console of the VM as root.
2. Set the DNS service to start on a reboot using the command:
   - `systemctl enable named`.
3. To change the forwarders, edit the file /etc/named.conf. Find the line labeled *forwarders*, and replace the forwarder 8.8.8.8 with your local DNS server IP addresses. This is only necessary if you want to be able to translate hostnames on your private network to IP addresses.
4. To configure the IP addresses, edit the file /var/named/weathervane.forward.zone.
   a. You will see that all of the hostnames in the Weathervane domain currently point to 127.0.0.1. Change these entries to have the IP address that you have assigned the hosts running the various services. If you have multiple services running on the same host, then all of the associated hostnames should point to the same IP address.
   b. You may need to add entries if you are using more instances of a given service than were included in the original file or are using multiple workloads or application instances.
   c. If using virtual IP addresses (not the default), then there must be at least as many IP addresses assigned to the www hostname as there are front-tier servers. For proper load-balancing, there should be the same number. This means that if you change the configuration you will need to adjust the number of IP addresses assigned to www and restart the named service. To change the number of addresses simply repeat the line for the www hostname multiple times with different IP addresses. The IP addresses that are assigned to www are virtual IP addresses. They must not be assigned to any physical NIC on any server. You enable the use of virtual IP addresses by setting *“useVirtualIp” : true,* in your configuration file. If you don’t have a compelling reason to use virtual IP addresses, it is easiest to leave useVirtualIP set to false.

d. If you want to use the vSphere power-control feature, or data-collection from vSphere ESXi hosts, and your entire Weathervane deployment is located on a private network, then you will need to add the IP addresses for your ESXi hosts. If using the convention-based hostnames discussed in Section **Error! Reference source not found.**, then these should use the hostname pattern AuctionVi1, etc. Note that it is also possible to explicitly define the host names for your VI hosts, using the viHosts parameter, in which case you will have to define the IP addresses for these ESXi hostnames. If running on a private network, your ESXi hosts will need to have interfaces configured on that network, and you will need DNS entries for those hosts. For example, in VMware vSphere you would need DNS entries for the IP addresses of the ssh-enabled vmkernel interfaces for the ESXi hosts on which you will be running the VMs.

e. Save the file.

5. From a shell prompt, restart the DNS server: `systemctl restart named`.

When creating the hosts you should have set the primary DNS server to be the IP address of the DNS server host. If you did not, you will need to edit /etc/resolve.conf to point to the DNS server VM on all Weathervane hosts. The DNS Server host should be the only nameserver listed in this file.

7.5 Cloning the Weathervane VM

Overview

In order to create the Weathervane deployment, you need a Weathervane host for each host in your deployment. If running in a virtual infrastructure, the easiest way to create these hosts is to clone the Weathervane host once for each host. The Weathervane host contains all software needed to act as any tier in a deployment. The run harness handles the configuration of each VM to run the particular services by using the predefined hostname to service mapping and the hostname to IP address assignments you create in the DNS server.

For each clone of the Weathervane VM, you need to configure the following:

- The IP address of the VM. The VM should be assigned a static IP address. This address must be the IP address that will map to the hostnames for the services you want to run on the VM. For example, if you want to run the database and message server together on a VM, you should assign the hostnames AuctionDb1 and AuctionMsg1 to the same IP address, and statically assign this IP address to a VM.
- The subnet mask for the network interface: This should be set as appropriate for your network.
- The gateway for the network interface: If the workload driver is running on the same network as all of the application VMs, then this should be set to

the IP address of the primary driver (AuctionDriver1). If using an isolated application network (see Section 7.10 for a description), then this should be set to the IP address for the application front-end host.

- The OS hostname of the VM: This should be set to the hostname of the service running on the VM, e.g. AuctionApp1. If multiple services will be running on the same VM, you can use any of the hostnames. The run harness can detect this condition and operate appropriately.
- The VM name of the VM: This should be set to the hostname of the service running on the VM, e.g. AuctionApp1. If multiple services will be running on the same VM, you can use any of the hostnames. The run harness can detect this condition and operate appropriately. Using a particular name for the VM is only necessary if using the power-control options, which are currently only supported on vSphere.
- The DNS domain name for the VM should be set to Weathervane.
- The DNS server for the VM should be set to the IP address of the VM running the DNS server. Typically, this will be the run-harness host.

Cloning using VMware vSphere

For VMware vSphere, the easiest way to configure the VMs for a Weathervane deployment is to clone the Weathervane VM using a vCenter customization specification. This will automate the entire configuration discussed above.

You should use the following settings for your customization specification:

- Target VM Operating System: Linux
- Computer Name: Use the virtual machine name.

  This will allow the run-script to find VMs and power them on and off as needed. The default VM names are given in Section 5.6, although these can be changes by changing variables in the run-script.
- Domain Name: Weathervane
- Time Zone: Use the appropriate setting for your location
- Network: Manually Select Custom Settings
  - Nic 1: Prompt User for Address. Use Subnet Mask appropriate for your subnet. For the Gateway, use the IP address of the primary driver or the application front-end VM, depending on whether you are using a connected or isolated application network.
- Primary DNS: Use the IP address that will be assigned to the DNS server VM.
- DNS Search Path: Weathervane

Create the VMs by cloning the Weathervane VM using the customization specification. Be sure to give the appropriate VM names as discussed in Section 8.4.

If you are planning to add additional NICs to a VM, such as for access to an external network, you should create the VM clone first, and then add the additional NIC.

7.6 Configuring Password-less ssh

In order for the run script to manage the Weathervane services and collect performance statistics, password-less ssh must be configured between the workload-driver system and all of the other VMs and ESXi hosts in the test-bed. The Weathervane host is set up so that when it is cloned, the driver VM will have password-less ssh access to its clones. However, for some virtual-infrastructure implementations you will also need to set up password-less ssh to the VI hosts in order to use the power-control or VI data-collection capabilities of the Weathervane run-harness.

Password-less ssh to ESXi Hosts

The run script uses password-less ssh to ESXi hosts in order to collect esxtop data, and in order to shut-down and power-on VMs. The ability to control the power of VMs allows automated execution of multiple run in which the number of VMs is varied. The run script will still function without these capabilities if password-less ssh is not configured to the ESXi hosts.

Configuring password-less ssh to ESXi hosts is performed as follows:

1. Enable the ESXi Shell and SSH on each ESXi host. This can be done in vCenter from the Configuration tab by editing the Services properties. It can also be done by connecting to the console of the ESXi host.
2. Using scp, copy the public key of the root user of the workload driver VM onto the ESXi host. Add the contents of this file on to the end of a file called authorized_keys in the /etc/ssh/keys-root/ directory of the ESXi host. Create this file if it does not exist.

Once you have configured password-less ssh to a host, you should try to ssh to that host from the command-line. This will test that the configuration was correct. It will also add a line to the file /root/.ssh/known_hosts for the remote host. If you do not do this before executing the run-script, the run will get stuck when the script encounters a prompt for a yes/no answer when trying to ssh/scp to the remote host. You should also ssh to each of the VMs from the workload-driver host in order to ensure that the known_hosts file is configured correctly for that VM.

7.7 Configuring a Time Source

In order for the Weathervane benchmark to work properly, the clocks on all of the VMs used in a Weathervane deployment should be kept synchronized. After running the autosetup script, the Weathervane VM will have the Network Time

Protocol (ntp) enabled ans set to use time sources available on the internet. This should be sufficient for most cases. However, you can use any time synchronization method that is appropriate for your test-bed.

To change the time-source for ntp, edit the file /etc/ntp.conf on each VM. Change the hostname in the line starting with *server*. If you set the parameter *harnessHostNtpServer* to true, the run harness will automatically configure your VMs to use the harness VM as their time source. This will ensure that time is synchronized, although not necessarily that the time is correct. For that to occur you must ensure that the time on your harness host is correct. Using the harness host as the time source is useful when your VMs are on an isolated network and do not have access to an external time source. When setting *harnessHostNtpServer* to true, you should also set *restartNtp* to true. This parameter will cause the harness to restart the NTP daemon on each VM before each run, which ensure that the time is properly synced. You can use this parameter even if you are using a different time source to ensure that the time on your VMs remains in sync.

To use a different time-synchronization method, for example VI-specific methods such as VMware tools, you must stop the ntp daemon on the VMs. To do this issue the commands `*systemctl stop ntpd*`, which will stop ntp, and `*systemctl disable ntpd*`, which will prevent it from starting after the next reboot.

7.8 Adding Additional Disks

In order to run Weathervane with more than a minimal number of users, it will be necessary to add additional data disks to the MongoDB and, if using, the file server VMs. See Section 9.4 for the space requirements. Adding disks for the database VM may improve performance, and will be necessary for very large numbers of users. The locations for mounting these additional disks is discussed in Section 9.5. If you will be performing a large number of runs for which you will want to keep the output data, you should also add a disk mounted on /root/weathervane/output on the run-harness host in order to increase the available storage space.

The steps involved in adding additional disks are as follows:

1. Make sure that all services are stopped and VMs are prepared for adding the new disks.
    a. If adding disks to the database VM
        i. Run the command `service mysql stop` on the database VM
        ii. Run the command `service postgresql-9.3 stop` on the database VM
    b. If adding disks to the MongoDB VM
        i. Run the command `mongod –f /etc/mongod.conf --shutdown ` on the MongoDB VM
    c. If adding disks for the file server

i. Run the command `umount /mnt/imageStore` on all powered-on driver, app server, and web server VMs.

ii. Run the command `service nfs stop` on the file server VM.

2. Prepare the mount locations.
    a. If you are adding a new disk for database data or logs you will need to move the contents of the /mnt/dbData or /mnt/dbLogs directories into a different location. These instructions will assume that the data was moved to /root/tmp/dbData or /root/tmp/dbLogs (e.g. `mv /mnt/dbData/* /root/tmp/data/.`). Do not use /tmp as the data may not persist across reboots.
    b. If adding new disks for the MongoDB VM(s), clear out the old contents of the directories.
        i. `\rm –rf /mnt/mongoData/*`
    c. If adding new disks for the File Server VM, clear out the old contents of the directories.
        i. `\rm –rf /mnt/imageStore/*`
3. Power off the VM and add additional disks.
4. Power the VM back on and log into the console as root.
5. The next step is to create a partition on the new disk. You can locate the new disk by examining the output of the command `fdisk –l`. Look for disks of the correct size with no partitions. The disks will likely be called /dev/sdb, /dev/sdc, etc.
    a. For each disk, create one partition which uses the entire disk. The commands are as follows:
        i. fdisk /dev/sdb (or the appropriate disk name)
        ii. n (new partition)
        iii. p (primary partition)
        iv. 1 (partition 1)
        v. Accept the defaults for First and Last cylinder
        vi. w (write the changes and exit)
6. Format the file system on each disk. Give the file system a label to make it easier to mount. To simplify the process, the file /etc/fstab on the Weathervane VM already has commented-out lines for mounting disks using the following labels: dbData, dbLogs,, mongoData, imageStore. These would be used as follows when creating the file system:
    a. mkfs.ext4 –L dbData /dev/sdb1
7. Edit /etc/fstab to add the lines to mount the new disks on the appropriate directory under /mnt.

a. The /etc/fstab file in the Weathervane VM has lines already configured to mount the disks on the appropriate locations. If you used the default directories and labels, then you can simply uncomment the correct lines.

8. Mount the disks with the command `mount –a`. The disks will be mounted on every subsequent reboot.
9. If adding disks for the database VM, move the database data and logs back onto the disks. For example `mv /root/tmp/dbData/* /mnt/dbData/.` or `mv /root/tmp/dbLogs/* /logs/.`
10. If adding disks for the database VM, change the permissions on the /mnt/dbData and /mnt/dbLogs directories to 777. This is necessary so that both MySQL and vPostgres can access directories under these mount points.
    a. `chmod 777 /mnt/dbData`
    b. `chmod 777 /mnt/dbLogs`

You are now ready to run the benchmark with the new disks.

7.9 Running Weathervane on a Private Network

There are a few special considerations when running Weathervane on a private network with no access to an external network or to the internet.

- The DNS configuration must not point to forwarders on an external network. Edit the file /etc/named.conf on the DNS host and remove the lines from the *options* blocks that begin with *forward* and *forwarders*.
- If you wish to use the power-control capabilities of the Weathervane run script, which automatically powers VMs on or off as needed on a vSphere VI, or to be able to run at log level 4, which enables collection of esxtop data from ESXi hosts, you will need to create vmkernel ports on each ESXi host using the NIC that is connected to the private network. The steps involved are as follows:
  1. Select a static IP address on the local network for each ESXi host.
  2. In the DNS zone file */var/named/Weathervane.forward.zone* on the DNS host, create an A record mapping a unique hostname for each ESXi host to the selected IP address. The default zone file contains some A records for this purpose using the hostnames Auctionesxhost1, etc.
  3. Using vCenter, on each ESXi host add a VMkernel Network Adapter to the vSwitch which is connected to the NIC for the private network. Set the IP address for this adapter to be that selected in the previous steps.

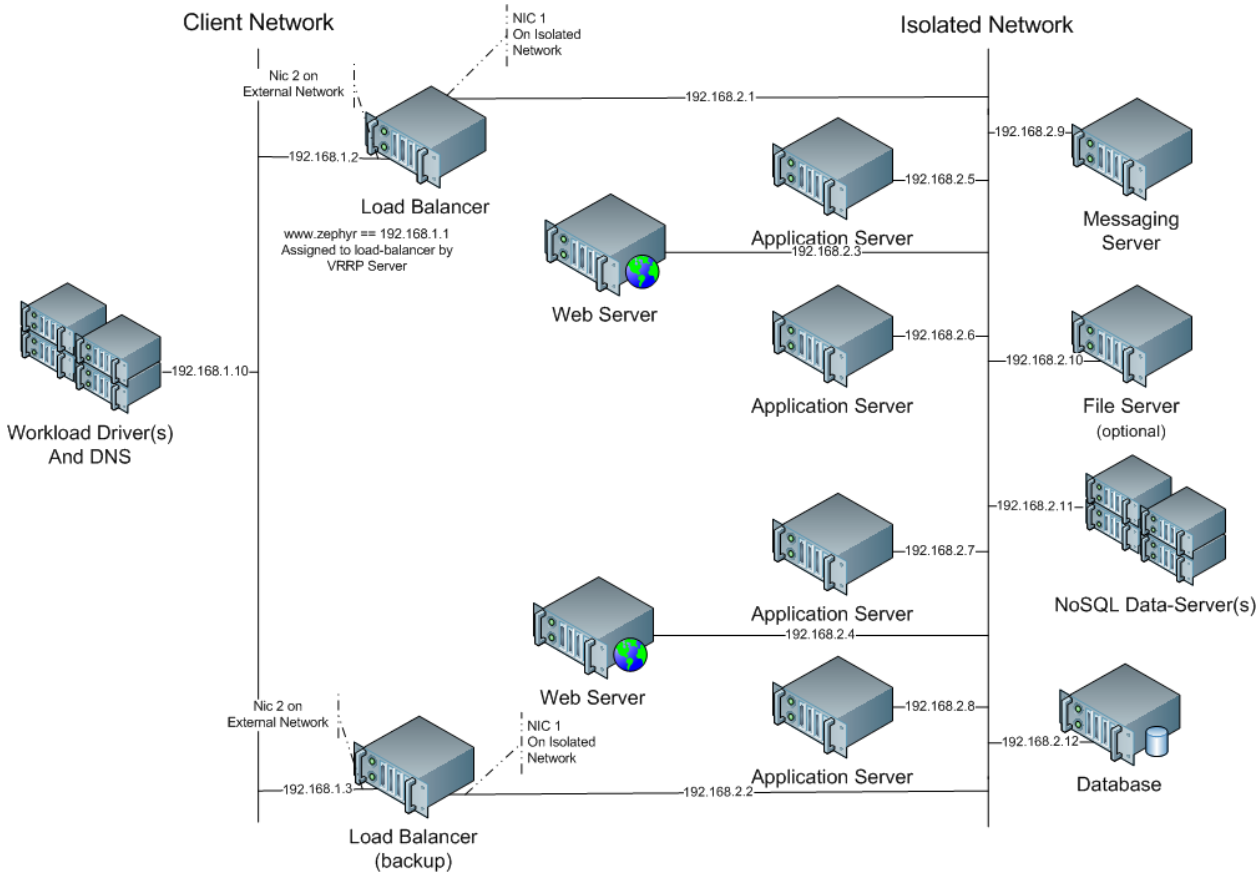
- Working with Weathervane on a private network will be easier if you have a NIC on the run-harness host which is connected to an external network. This will simplify moving data off of the workload driver, and will allow you to connect to the testbed using ssh or VNC. Otherwise you will be restricted to interacting with the VMs through the VI-provided console.

7.10 Running Weathervane with an Isolated Application Network

It is possible to run Weathervane in a configuration where the application services are running in VMs on a private subnet that is not directly accessible to the workload drivers. This configuration would be useful when running the application VMs on a VI host with an internal-only network that has no associated physical NIC. It could also be used when the application components are running on a private network in one location, such as in a cloud deployment, and the workload drivers are running in a different location.

In order to use Weathervane with an isolated application network, the VM acting as the front-end for the application must have two NICs, one on the isolated network, and one on a network that is accessible to the workload drivers, which we shall refer to as the client network. Figure 4 shows a full Weathervane deployment with all of the application components running on a network that is isolated from the workload drivers. In this configuration, the load-balancer is the front-end for the application. It would need two NICs, one on the isolated network and another on the external network. Each NIC would be assigned an IP address for the network to which it is attached. The IP addresses shown are just to clarify the separate networks, and you should use appropriate addresses for your networks. If the deployment configuration included a single web server and no load-balancer then the web server would be the front-end and would need to have two NICs. Likewise, in a deployment with a single application server acting as the front end, the application server would need two NICs. In all cases the VRRP server will handle the assignment of the IP address for www.Weathervane to the front-end server.

Figure 4 Example Weathervane Deployment with Isolated Application Network



In order to run Weathervane in this configuration additional configuration steps are must be performed these are as follows.

1. A second NIC must be added to the front-end VM for the Auction application. One NIC should be configured for the external network, and the other for the isolated network. In the example from Figure 4 the NIC on the external network has the IP address 192.168.1.2, and the NIC on the isolated network has the IP address 192.168.2.1.
2. The front-end VM should have IP packet forwarding enabled in the OS. This can be done by placing the line *"net.ipv4.ip_forward = 1"* in the file /etc/sysctl.conf, and reloading the sysctl configuration with the command `*sysctl –p*`. Note that configuration step has already been performed on the Weathervane VM.
3. Edit the DNS zone file as described in Section 0. The configuration should be as follows:
    a. The hostnames for the workload drivers should map to their addresses on the client network.
    b. The hostname for the front-end server should map to its IP address on the client network. For example, in Figure 4 the DNS entry for AuctionLb1 would map to 192.168.1.2.

c. The hostnames for all other VMs on the isolated network should map to addresses on the isolated network. For example, in Figure 4 the DNS entry for AuctionApp1 would map to 192.168.2.5.

4. The primary workload-driver must have a route defined to the isolated network through the front-end server.
    a. On the primary driver, use ifconfig to determine the name for the NIC that is connected to the client network. If you are running Weathervane on a private network, you may have two NICs on the primary driver, one attached to the client network, and one attached to an externally accessible network. The name will probably be of the form eth0 or eth1. The remainder of these instructions will assume that the name is eth0.
    b. Edit the file /etc/sysconfig/network-scripts/route-eth0. Create the file if it does not exist. Add a line of the form:

        192.168.2.0/24 via 192.168.1.2 dev eth0

        Replacing the following as appropriate for your network:

        - Replace *192.168.2.0/24* with the base network address and number of significant bits in the netmask for the isolated network.
        - Replace *192.168.1.2* with the IP address of the NIC from the application front-end server that is on the client network.
        - Replace *eth0* with the name of the NIC from step a.

    c. Restart the network services with the command `*service network restart*`. At this point you should be able to ssh from the primary driver to VMs located on the isolated network.

5. All servers running on the isolated network must be configured to have the front-end server act as their primary network gateway.
    a. If using VMware vSphere, you can do this by specifying the IP address of the front-end server on the isolated network as the primary gateway in the customization specification used to create the VMs that are located on the isolated network.
    b. If manually editing the network configuration:
        i. First find the name of the NIC that is connected to the isolated network. We will assume that it is eth0.
        ii. Edit the file /etc/sysconfig/network-scripts/route-eth0. Make sure that it contains only the line: "*default via 192.168.2.1*", where you should replace *192.168.2.1* with the IP address of the front-end server on the isolated network.

iii. Restart the network services with the command `*service network restart*`. At this point you should be able to ssh from VMs located on the isolated network to the primary driver.

Once this configuration is complete, you will be able to run Weathervane with the application services running on the isolated network.

8 Weathervane Configuration File

8.1 Introduction

The Weathervane configuration file is the bridge between the physical deployment of hosts and services and a run of Weathervane. It contains the values for parameters that describe the desired deployment to the run harness. It also contains run-specific parameters such as the number of simulated users to use, the run mode to use, and tuning parameters for the various services.

The run harness always reads a configuration file at the start of execution. It is also possible to specify parameters as command-line options to the run harness. Parameters specified on the command-line override the same parameter specified in the configuration file. Using command-line parameters in this way can simplify performing multiple runs in which only a small number of parameters vary among the runs.

8.2 The Configuration File

At start-up, the Weathervane run-script reads parameters from a configuration file. The Weathervane parameter file uses the JSON data format to describe the parameter name/value pairs and, in more complex deployments, the hierarchical structure of the workload, application instance, and host configurations. A brief primer on JSON is given in the next section.

The default configuration file is /root/weathervane/weathervane.config. The Weathervane host comes with a version of this file that contains the most commonly used Weathervane parameters along with a brief description of their purpose. If a parameter is not specified in the configuration file or on the command-line, then Weathervane uses a default value. The default values are described in the original configuration file, and can also be found in the descriptions obtained by running the Weathervane run-script with the --help option.

It is possible to use a file other than /root/weathervane/weathervane.config as the configuration file. To do this, simply specify the file with the *--configFile filename* option to the run-script. For example */root/weathervane/weathervane.pl --configFile /root/weathervane/config2* will use the file /root/weathervane/config2 as the configuration file.

8.3 JSON Primer

The Weathervane configuration file uses an extended JSON format. Quoting from the json.org web site: "**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate."

The basic construct in JSON is a JSON object, which is represented as a set of key/value pairs enclosed in curly braces. The entire Weathervane configuration file is a single JSON object.

All keys in a JSON object are strings, which must be enclosed in double quotes.

Values in JSON may be strings enclosed in double quotes, numbers, the Boolean values true and false (no double quotes), a JSON object, a JSON array, or the null value. A JSON array is a sequence of JSON values surrounded by square brackets and separated by commas. The null value is not used in the Weathervane configuration file.

The Weathervane run harness allows two extensions to the JSON standard to be used in the configuration file:

- The file may contain comments, which start with the # symbol. All comments are ignored when parsing the file.
- The last key/value pair in a JSON object, and the last value in a JSON array, may be followed by a comma. This simplifies making changes to the file as you can delete or comment out lines and not worry about needing to remove the comma on the previous line.

Therefore, the basic definition of parameters in the Weathervane configuration file will look like:

```
{
      "users" : 500,
      "description" : "Initial Test Run",
      "reloadDb" : false,
}
```

This shows the definition of parameters which have a number, string, and Boolean value. In more complex configuration files, discussed in Section 8.6, the parameter value may be a JSON object or array.

8.4 Basic Configuration Descriptions

For most Weathervane deployments, the configuration file can be quite simple. A basic Weathervane configuration file is a JSON object containing a set of key/value pairs in which the parameter values are all numbers, strings, or Boolean values. Weathervane also supports an explicit, hierarchical, configuration style in which parameter values can be specified for each service independently.

The key differences between the basic and explicit configuration are:

- In the basic configuration, the number of each type of service is specified with a single parameter, e.g. *numAppServers : 3*. Using the hierarchical configuration, a separate JSON object is used to specify the configuration of each individual service.
- In the basic configuration, parameters are specified once, and apply to all relevant services. In the hierarchical configuration, it is possible to specify different values of parameters for each service instance.

The basic format will be sufficient for most deployments. However, there are some drawbacks to using the basic configuration file format:

- When using the basic configuration, it is not possible to specify different parameters for each instance of a service type in an application instance. For example, all application servers in an application instance must use the same JVM parameters.
- When using Docker, all services must run in Docker containers. With hierarchical configuration, it is possible to run only specific service types or even individual service instances in Docker containers.
- The basic configuration requires the use of the convention-based naming discussed in Section 8.5. As a result, it is necessary to run a DNS server to translate the host names to IP addresses. This is actually not a significant burden, as the Weathervane auto-setup script configures a DNS server for the Weathervane domain. You will only need to edit the Weathervane zone file as needed and configure the DNS server to start on boot on your primary node. If multiple services are running on a single VM then the hostnames associated with those services must all translate to the same IP address.
- If using virtual infrastructure services, you must give your VMs the same name as the hostname used for one of the services that will run on that VM.

8.5 Convention-based Host Name Assignment

All of the services in a Weathervane run, including workload drivers and data managers, must be mapped to the host on which they will be executed. Multiple services may be run on a given host, or all services may be executed on separate hosts. It is possible in the Weathervane configuration file to explicitly specify the valid hosts and the mapping between services and hosts, as will be covered in Section 8.6. However, this can be cumbersome for large deployments. As a result, the Weathervane run harness provides a way to specify configurations using pre-defined conventions for host and VM names.

This section describes the conventions used for mapping services to hostnames for the most common case of a single workload and application instance. The use of multiple workloads and application instances, along with the extended conventions for these deployments, are discussed in Section 11. Note that if you are planning to run configurations with multiple workloads or application

instances, you should use the extended conventions even in your initial set-up. This will simplify the task of adding additional workloads later.

When using convention-based naming, the configuration of services to be used in a run is specified by defining, in the Weathervane configuration file, the number of each type of service to be used. The mapping between these services and hosts is defined by configuring a DNS server to define the mapping between the hostname associated with each service and the IP address of the host on which it will run.

For example, consider a Weathervane deployment with 3 workload drivers, and a deployment of the Auction application which uses 1 web server, 3 application servers, 1 coordination server, 1 message server, 1 database node, 1 NoSQL data-store node, and 1 configuration manager. The complete Weathervane configuration file which defines this configuration would look like:

```
{
      “numWorkloads” : 1,
      “numAppInstances” : 1,
      “numDrivers” : 3,
      “numWebServers” : 2,
      “numAppServers” : 3,
      “numCoordinationServers” : 1,
      “numMsgServers” : 1,
      “numDbServers” : 1,
      “numNosqlServers” : 1,
      “numConfigurationManagers’ : 1,
}
```

In this instance, all of the other parameters for the run(s) would take their default values. Details about the available parameters for a run of Weathervane are given in Section 8.7. Note that the *numWorkloads* and *numAppInstances* parameters must be specified and set to 1.

The lines in the DNS zone configuration file to map the services to the IP address of the host on which they should run would look as follows. The DNS configuration is described in more detail in Section 0.

```
AuctionDriver1 IN A 192.168.1.1

AuctionDriver2 IN A 192.168.1.2

AuctionDriver3 IN A 192.168.1.3

AuctionWeb1 IN A 192.168.1.4

AuctionApp1 IN A 192.168.1.5

AuctionApp2 IN A 192.168.1.6

AuctionApp3 IN A 192.168.1.7

AuctionCs1 IN A 192.168.1.8
```

```
AuctionMsg1 IN A 192.168.1.9
AuctionDb1 IN A 192.168.1.10
AuctionDm1 IN A 192.168.1.10
AuctionNosql1 IN A 192.168.1.9
AuctionCm1 IN A 192.168.1.8
```

The only additional configuration required is to create Weathervane hosts which use these IP addresses. The Weathervane run harness will they handle the tasks related to starting the services on the appropriate hosts. Note that in the example above, the message service and the NoSQL data store will run on the same host, the coordination service and configuration manager will run on the same host, and the data manager will run on the same host as the database.

For each service, the host name associated with a service is generated using the following pattern, where a '+' denotes string concatenation:

```
Hostname = hostnamePrefix +
                serviceSuffix + serviceInstanceNum
```

Where each of these components has the following default values:

- **hostnamePrefix**: "Auction"
- ***service*Suffix:** There is a different suffix defined for each type of service supported by the Weathervane run harness. The values are:
    - **workloadDriverSufffix:** "Driver"
    - **dataManagerSuffix:** "Dm"
    - **lbServerSuffix:** "Lb"
    - **webServerSuffix:** "web"
    - **appServerSuffix:** "app"
    - **msgServerSuffix:** "msg"
    - **dbServerSuffix:** "Db"
    - **nosqlServerSuffix:** "Nosql"
    - **fileServiceSuffix:** "File"
    - **configurationManagerSuffix:** "Cm"
    - **coordinationServerSuffix:** "Cs"
- **serviceInstanceNum:** Numbers from 1 to p, where p is the number of instances of a specific service in the workload or application instance.

All of the prefix and suffix defaults can be overridden in the Weathervane configuration file. They can be overridden for each application instance, for all application instances in a given workload, or for all workloads.

The run harness also defines conventions for host names for virtual infrastructure hosts. The pattern is similar to that used for service host names.

```
VIHostname = hostnamePrefix + viHostSuffix + instanceNum
```

Where each of these components has the following default values:

- **hostnamePrefix**: “Auction”
- **viHostSuffix**: “viHost”
- **instanceNum**: Numbers from 1 to n, where n is the number of VI Hosts

As with service host names, VI host names need to be mapped to their IP addresses in the DNS zone file.

There is one additional hostname, referred to as the *www* hostname, that must be configured in DNS when running with Virtual IP addresses for your front-tier servers. *Note that using virtual IP addresses is not the default, so you can ignore this information if you wish*. Virtual IP addresses are IP addresses which are not physically tied to a particular host, but are allowed to float among the front-tier servers. The assignment of the virtual IP addresses to particular front-end servers is managed by an IP management service, VRRP, using keepalived in the case of Weathervane. When virtual IP addresses are used, the workload drivers access the front-tier servers using the www hostname rather than the hostnames of the front-tier services. The front tier is the first among the load-balancer, web-server, or application-server tiers, in that order, to be included in your configuration.

When using virtual IP addresses, you assign IP addresses to the www hostname that are not assigned to any physical NIC on any server. However, they must be valid IP addresses on the subnet connected to the front tier of servers that is exposed to the workload driver. In the DNS zone file there must be at least as many virtual IP addresses assigned to the www hostname as there are first-tier servers. For proper load balancing there should be the same number. The run harness will handle assigning these IP addresses to the proper VMs using the VRRP server. When using virtual IP addresses, you can change which service is your front tier without changing the DNS configuration, but only if you have the same number of servers in your new front tier. Using virtual IP addresses also ensures that if one front-tier server goes down, the www hostname, and thus the application, is still accessible on all assigned IP addresses. This improves the availability of the deployment, and thus more representative of a production deployment. However, for a run with no failures, there is no performance impact to the choice of virtual or physical www IP addresses. You enable the use of virtual IP addresses by setting *“useVirtualIp” : true,* in your configuration file.

When using only a single workload and application instance, the www hostname is simply *www*.

8.6 Explicit Configuration and Naming

While using convention-based naming simplifies specifying a Weathervane deployment, there are times when you will need to specify different names or override defaults in way that cannot be handled using the convention-based configuration style. For these cases, Weathervane supports a more expansive specification style that allows explicit specification of all parameters for each host and service instance.

The explicit configuration style replaces the simple specification of the number of each type of service with a JSON array containing a set of JSON objects, each of which gives the parameters for a single instance of that service. For example, the following is a configuration file that uses the explicit style to specify exactly the same configuration described in Section 8.4.

```
{
      “numWorkloads” : 1,
      “numAppInstances” : 1,
      “drivers” : [ {}, {}, {} ],
      “webServers” : [ {}, {}, ],
      “appServers” : [ {}, {}, {}, ],
      “coordinationServers” : [ {}, ],
      “msgServers” : [ {}, ],
      “dbServers” : [ {}, ],
      “nosqlServers” : [ {}, ],
      “configurationManagers’ : [{}, ],
}
```

In this case, no parameters are specified for the services instances. This means that all will use the default parameter values, including the default, convention-based, hostnames.

Once the configuration is specified in this way, it is possible to override the defaults for each individual service. For example, the following configuration file would cause the database service to use MySQL rather than the default PostgreSQL, and gives explicit hostnames for the application servers. Note that a DNS translation must exist for all hostname specified this way.

```
{
      “numWorkloads” : 1,
      “numAppInstances” : 1,
      “drivers” : [ {}, {}, {} ],
      “webServers” : [ {}, {}, ],
      “appServers” : [ { “hostName” : “AppHost1”, },
                             {“hostName” : “AppHost2”, },
                             {“hostName” : “AppHost3”, }, ],
      “coordinationServers” : [ {}, ],
      “msgServers” : [ {}, ],
      “dbServers” : [ { “dbServerImpl” : “mysql”, }, ],
```

```
        “nosqlServers” : [ {}, ],
        “configurationManagers’ : [{}, ],
}
```

It is also possible to mix the convention-based and explicit notations. For example, the following specifies the same configuration as the previous specification.

```
{
        “numWorkloads” : 1,
        “numAppInstances” : 1,
        “numDrivers” : 3,
        “numWebServers” : 2
        “appServers” : [ { “hostName” : “AppHost1”, },
                         {“hostName” : “AppHost2”, },
                         {“hostName” : “AppHost3”, }, ],
        “numCoordinationServers” : 1,
        “numMsgServers” : 1,
        “dbServers” : [ { “dbServerImpl” : “mysql”, }, ],
        “numNosqlServers” : 1,
        “numConfigurationManagers’ : 1,
}
```

The Weathervane configuration file also has a hierarchical aspect. Parameters that are specified at the top level of the configuration file will override defaults for services specified at a lower level, such as in an “appServers” array. For example, in the following configuration file the specification of the “hostName” and “useDocker” parameters at the top level will cause all of the services to run in Docker containers on the same host, *AuctionDockerHost*, except the workload drivers which explicitly override the hostname parameter to map to hosts *DriverHost1*, *DriverHost2*, *DriverHost3*.

```
{
        “hostName” : “AuctionDockerHost”,
        “useDocker” : true,
        “numWorkloads” : 1,
        “numAppInstances” : 1,
        “drivers” : [ {“hostname” : “DriverHost1”},
                      {“hostname” : “DriverHost2”},
                      {“hostname” : “DriverHost3”} ],
        “webServers” : [ {}, {}, ],
        “appServers” : [ {}, {}, {}, ],
        “coordinationServers” : [ {}, ],
        “msgServers” : [ {}, ],
        “dbServers” : [ {}, ],
        “nosqlServers” : [ {}, ],
        “configurationManagers’ : [{}, ],
```

```
}
```

The hierarchical nature of configuration files is particularly relevant to the deployments with multiple application instances, which are discussed in Section 11.

8.7 Configuration Parameters

Introduction

This section discusses the parameters that can be specified in the Weathervane configuration file or on the command-line. Many of these parameters will only be needed for more advanced configurations, and so it will be best to skim this section on first reading and then use it as a reference as you explore the more advanced features of Weathervane.

Command-Line Parameters

Parameters can be specified at the run-harness command-line as follows:

$ */root/weathervane/rweathervane.pl* –paramName[=value] …

where *paramName* is the name of the parameter, and *value* is the value. The parameter name is the same as the name for the same parameter in the configuration file. Not all parameters require values. In particular, options that take Boolean, i.e. true/false, values in the configuration file are specified without values. Simply including the parameter on the command line, e.g. –loadDb, sets the associated parameter to true. To set a Boolean parameter to false, include it on the command-line with *–no* before the parameter name, e.g. –noloadDb.

Parameters specified on the command-line override the same parameter specified in the configuration file. This can be useful when performing multiple runs where only a small number of parameters are changed. For example, when performing multiple runs at different numbers of users, you can avoid having a separate configuration file for each run by specifying the number of users at the command-line using the –users option.

There is a small set of parameters that are generally only useful on the command-line. These are as follows:

- **--help**: This will print a list of the parameters for the Weathervane run
- **--stop:** This can be used to cleanly stop the current run, or to clean up after a run that failed midway through.

Scripting Multiple Runs

As discussed in the section on command-line parameters, it is often desirable to set all of the basic parameters in the configuration file and then override key parameters on the command-line. This process can be simplified even further by using a shell script to contain multiple invocations of the run-script. The file /root/weathervane/runmany.pl provides an example of this process. This file can

be used to perform multiple runs of the benchmark using a single configuration file.

Parameters

This section describes the parameters for the Weathervane run-script. The parameters are separated into five sections:

- General Run Parameters
- Deployment Configuration Parameters
- Docker Specific Parameters
- Run Mode Parameters
- Run Phase-Specific Parameters
- Component Tuning Parameters

These parameters are described in the following sections.

8.7.1.1 General run parameters

The general parameters control the basic execution of the run, including the number of users to drive against each instance and the duration of the run. The individual options are discussed below:

- **users**: This controls the number of simulated users that will be actively driven against the application by the workload generator.

  If you are running multiple workloads/application-instances (see Section 11), you can override the number of users for each application-instance from the command-line. The --users command-line parameter takes a list of values, e.g. --users=11000,2000,4000. The values in the list are applied to the application-instances in the order that they are defined in the configuration file. For example, if you have two workloads, the first with one app instance and the second with two, then the above command-line parameter would set workload1/appInstance1 to run with 11000 users, workload2/appInstance1 to run with 2000 users, and workload2/appInstance2 to run with 4000 users. If you specify fewer values than the number of appInstances, the remaining appInstances will use the users value from the configuration file, if specified, or the default. If you specify more values than appInstances then the extra values will be ignored.

  It is also possible to drive the Auction using a load profile in which the number of users varies during the run. This is discussed in Section **Error! Reference source not found.**.

  - Default: 300.

- **runLength**: The runLength parameter allows you to control the length of a run using pre-defined *short*, *medium*, and *long* key-words. These

keywords imply specific lengths for the duration of the ramp-up, steady-state, and ramp-down periods of a run.

  - Allowed values: *short* (120s, 180s, 60s), *medium* (720s, 900s, 60s), *long* (720s, 1800s, 120s).
  - Default: *medium*

- **rampUp**, **steadyState, rampDown**: These options are used to specify the duration of the ramp-up, steady-state, and ramp-down periods of a run. They override the values implied by the runLength parameter. The values defined by runLength are applied first, and then the specific values are used to override the defaults for that run-length.

The run-duration intervals (rampUp, steadyState, and rampDown) are used to control:

- The overall length of the run, which equals rampUp + steadyState + rampDown
- Which response-time samples are used when computing the pass/fail result of the run. Only operations which occur during the steadyState period are used for the QoS calculations.
- The period of time in which performance statistics are collected from the hosts and the various services when running with logLevel equal to 2 or higher. Performance statistics are only collected during the steady-state period.
  - Default: No default. The default runLength values are used.

- **description:** This parameter allows you to specify a text string that is used to provide information about the run. It is placed in a column in the weathervaneResults.csv file, and in a file in the output directory.
- **logLevel**: The logLevel parameter is used to specify how much data to collect during the run and to gather up and store in the output directory for the run. The levels go from 0 to 4. Each level also collects all of the data collected in all of the lower levels.
  - The data collected by each level is as follows:
    - 0: Only store the results file and logs generated by the run harness.
    - 1: After the run completes, collect the log files from all software services and store them in the output directory.
    - 2: Collect host-level (e.g. sar) performance data on all VMs.
    - 3: Collect service-specific performance data from all software services.

        - 4: Collect performance data from virtual infrastructure hosts. Only the collection of esxtop data from ESXi hosts in a vSphere VI is currently supported.
    - Default: 1

- **maxLogLines:** In order to avoid filling up the output directory when a run encounters a large number of errors, as may happen when some portion of the deployment is heavily overloaded, the run harness truncates all log files to be at most 4000 lines. The 4000 lines will include the first and last 2000 lines of each log file. You can change this limit by setting the value of maxLogLines in your configuration file.
- **showPeriodicOutput:** By default, the periodic performance statistics are sent only to the run.log file. The periodic stats show the average throughput and response-time, as well as the number of each operation failing the response-time QOS. To have this information echoed to the console, set this parameter to true in the configuration file or specify –showPeriodicOutput on the command-line. Note that if you are running multiple workloads, you can set this parameter in the hierarchical configuration separately for each workload. If you set it to true for multiple workloads, the output from both workloads will be intermingled on the console.
- **stopServices:** This parameter controls whether to stop the application services at the end of a run. The default is to stop the services. This parameter leaves the services running so that their state can be examined after a run. It is mainly useful for debugging purposes and will not be used by most users of Weathervane. To disable the stopping of the application services at the end of a run, use the command-line parameter —nostopServices, or set “stopServices” : false, in your configuration file.

8.7.1.2 Deployment Configuration Parameters

The deployment configuration parameters describe the aspects of the deployment that are needed by the Weathervane run-script to control the execution of the benchmark. These include the number of instances to use for each service that can be scaled horizontally, and the implementation to use for services, such as the database or web server, that support multiple implementations.

Note that the default for all numXXX parameters is 0, to allow for the use of explicit configuration. As a result, if using basic configuration a value must be set for all of the relevant parameters.

- **numWorkloads**: This is the number of workloads to be used in the run. When using basic configuration, this must be set to at least 1.
    - **Default**: 0.
    - It is also possible to specify the workloads using the workloads parameter in a hierarchical configuration (See Section 11.3).

- **numAppInstances**: This is the number of application instances to use for each workload. When using basic configuration, this must be set to at least 1.
    - **Default**: 0
    - It is also possible to specify the appInstances using the appInstances parameter in a hierarchical configuration (See Section 11.3).
- **numDrivers**: This controls the number of workload-driver nodes to be used to drive the load for the run. There must be at least one driver for each workload.
    - Default: 0
    - It is also possible to specify the number of workload drivers using the drivers parameter in the explicit configuration style.
- **numLbServers**: This controls the number of load-balancer server VMs to be used. The load-balancer sits in front of the web servers, or, if there are no web servers in the deployment, the application servers.
    - Default: 0
    - It is also possible to specify the number of load balancers using the lbServers parameter in the explicit configuration style.
- **numWebServers**: This controls the number of web server VMs to be used
    - Default: 0
    - It is also possible to specify the number of web servers using the webServers parameter in the explicit configuration style.
- **numAppServers**: This controls the number of application server VMs to be used.
    - Default: 0
    - It is also possible to specify the number of app servers using the appServers parameter in the explicit configuration style.
- **numCoordinationServers**: This controls the number of coordination servers to be used.
    - Default: 0
    - Allowed values: There must be either 1 or 3 coordination servers. However this value can be 0 if using the coordinationServers parameter in explicit configuration.
- **numConfigurationManagers**: This controls the number of configuration managers to be used.
    - Default: 0

    - Allowed values: There must be either 0 or 1 configuration managers. A configuration manager is required when using application elasticity (see Section 11.4) or when pre-warming of application servers is desired. Because pre-warming of app servers is the default (i.e. the default for the prewarmAppServers parameter is true), a configuration manager should be included in your configuration unless you explicitly set prewarmAppServers to false. It is also possible to specify the number of configuration managers using the configurationManagers parameter in the explicit configuration style.
- **numMsgServers**: This controls the number of message server VMs to be used.
    - Default: 0
    - Allowed values: There must be at least 1 message servers.
    - It is also possible to specify the number of message servers using the msgServers parameter in the explicit configuration style.
- **numNosqlServers**: This controls the number of NoSQL (MongoDB) nodes to be used during the run
    - Default: 0
    - Allowed values: There must be at least one NoSQL node.
    - It is also possible to specify the number of NoSQL servers using the nosqlServers parameter in the explicit configuration style.
- **nosqlSharded**: This controls whether the MongoDB deployment is to be sharded. If numNosqlServers is greater then 1, then either nosqlSharded or nosqlReplicated must be true.
    - Default: false
- **nosqlReplicated**: This controls whether the MongoDB deployment is to consist of a replica set. If numNosqlServers is greater then 1, then either nosqlSharded or nosqlReplicated must be true.
    - Default: false
- **numFileServers**: This controls the number of file server VMs to be used during the run.
    - Default: 1
    - Allowed values: 0 or 1
    - It is also possible to specify the file server using the fileServers parameter in the explicit configuration style.
- **lbServerImpl**: Controls which implementation to use for the load balancer.
    - Allowed values: haproxy

    - Default: haproxy
    - It is also possible to specify the load-balancers using the lbServers parameter in the explicit configuration style.
- **webServerImpl**: This allows the user to choose between Apache httpd 2.4, and nginx when running the benchmark.
    - Allowed values : httpd, nginx
    - Default: nginx
- **appServerImpl:** Controls which implementation to use for the application server.
    - Allowed values: tomcat
    - Default: tomcat
- **dbServerImpl**: This allows the user to choose between MySQL and PostgreSQL when running the benchmark.
    - Allowed values : mysql, postgresql
    - Default: postgresql
- **nosqlServerImpl**: Controls which implementation to use for the NoSQL data store.
    - Allowed values: mongodb
    - Default: mongodb
- **msgServerImpl**: Controls which implementation to use for the message server
    - Allowed values: rabbitmq
    - Default: rabbitmq
- **fileServerImpl**: Controls which implementation to use for the file server.
    - Allowed values: nfs
    - Default: nfs
- **imageStoreType**: Controls which image-store implementation to use. When set to *mongodb*, images for the Auction application are stored in MongoDB and no network filesystem is used. When set to *filesystem*, the images are stored in the network filesystem and the web server, if present, handles requests for images. When set to *memory*, images requests are handled by the app server, which always returns the same image. Image uploads are dropped by the app server. The memory mode is only included to help debug scalability issues, and does not represent a realistic mode of operation.
    - Allowed values : mongodb, filesystem, memory

- Default: mongodb

- **restartNtp:** Setting this parameter to true causes the run harness to restart ntp on all hosts at the beginning of each run. This ensures that the clocks are synced on all hosts, but is unnecessary if ntp is properly configured on the hosts. It is really only needed for situations where the non-harness hosts are on a private network and don't have access to a time source.
- **harnessHostNtpServer:** Setting this parameter to true cause the run harness to edit ntp.conf on all hosts at the beginning of each run to point to the run harness host as the time server. This helps ensures that all hosts have a common time source, but is unnecessary if ntp is properly configured on the hosts. It is really only needed for situations where the non-harness hosts are on a private network and don't have access to a time source.

8.7.1.3 Docker Parameters

All of the services used in an Auction deployment, except for the file server, can be run either directly on the OS or in a Docker container. More information about running Weathervane with Docker, along with a description of the relevant parameters, can be found in Section 10.

- **useDocker**: When set to true, the run harness will deploy the Weathervane services in Docker containers rather than directly on the OS. In a hierarchical configuration, this parameter can be set at the level of a workload, an application instance, a service type, or a specific service instance. When set at the top level, it applies to all services.
  - **Default**: false
- **dockerNamespace:** The dockerNamespace parameter tells the run harness where to find the Docker images for the Auction application. Building the images is discussed in Section 10. If the images are in your Docker Hub account, then dockerNamespace should be set to your Docker Hub username. If the images are in a private registry, then dockernamespace should be set to "*hostname:portNumber"*, where hostname is the name or IP address of the host on which the registry is running, and portNumber is the port on which the registry is listening.
  - **Default:** There is no default. This parameter must be set in order to use Docker.

8.7.1.4 Run Mode Parameters

The Weathervane run-harness provides a great deal of flexibility in specifying the type of workload to be used, the goal of each invocation of the harness, and the manner in which individual runs are to be performed. This allows it to be used in a wide variety of performance-evaluation and benchmarking scenarios. The parameters that control the behavior of the harness are the *runStrategy* and

runProcedure. Note that the behavior of these options for a run with multiple workloads and/or application-instances is discussed in Section 11.4.

- **runStrategy:** The run strategy selects the objective of an invocation of run harness. Currently implemented run strategies are:
    - single: This is the default run strategy. It causes the run harness to perform a single run and then exit.
    - findMax : Selecting this strategy causes the run harness to perform runs until it has found the maximum number of users that can be driven against the current application deployment while satisfying the passing criteria. See below for a complete description of this run strategy.
    - targetUtilization : This strategy causes the run harness to perform multiple runs, adjusting the number of users in each run, in order to find the number of users that drives the average CPU utilization of a particular service tier to a specified target. See below for a description this run strategy.
- **runProcedure:** The run procedure selects among various options for how individual runs are to be performed. At present, the findMax and targetUtilization runStrategies require the use of the "full" run-procedure. The other options ("prepareOnly", “runOnly”, and "stop") can only be used with the single run strategy. The run procedures that can be specified are:
    - full: This runProcedure performs a complete run of Weathervane each time it is invoked by the selected runStrategy.
    - loadOnly: This run procedure loads the data into the data services, but does not start the application or run the workload driver. It is useful for preparing the data for future runs.
    - prepareOnly: This run-procedure prepares all of the application services, starts the Auction application, and then exits. It does not start the workload-driver. This runProcedure can be useful when debugging configuration and deployment issues. Once the application is running, you can manually connect to the application using a web browser and examine the state of the services to identify issues.
    - runOnly: This run procedure runs the workload driver and performs all of the post-run steps of collecting data and shutting down services. When you use this run procedure the data must already be prepared and the services running. It is typically used after the prepareOnly run procedure with the same configuration file in situations where you want a delay between run preparation and actually starting the workload. You must include the delay time between preparation and the runOnly run procedure in your maxDuration setting.

- stop: The stop run-procedure is used to stop a run that is already in progress. Invoked from a separate command-line, this will cleanly stop the workload and shut down all of the application services. It should also be used after running with the prepareOnly run-procedure to ensure that all services are properly cleaned up before a new run. For ease-of-use, the run harness also provides a --stop command-line parameter, which is shorthand for invoking this runProcedure. Note that stop should be invoked with the same configuration file and command-line parameters used when a run was started in order to ensure all services are properly shut-down.

The Maximum Finding run-strategy (findMax) works as follows:

1. The run harness performs a run at the given number of users, as specified by the users parameter.
2. If the run passes the benchmark criteria, the number of users is increased by a given amount, otherwise the number of users is decreased by a given amount (but not below a specified number of users). The benchmark is re-run at this new value.
3. The pass/fail status is checked after each run, with the number of users changed after each run. Every time the sign of the increase changes, the change in the number of users is halved.
4. Once the change in the number of users falls below a given threshold, the run script stops and a maximum is declared.

The following parameters are used to control the maximum finding mode:

- **initialRateStep:** This is the size of the increment by which the number of users is initially increased or decreased. This value will be halved each time the direction of change reverses.
  - Default: 1000
- **minRateStep:** This is the minimum size of the change in the number of users. Once a passing run is found which is within this increment of a passing run below, and a failing run above, the run script will declare that run to be the maximum.
  - Default: 125
- **repeatsAtMax**: Setting this parameter to a value greater than 0 causes the run harness to repeat the findMax procedure the indicated number of times. For each subsequent findMax iteration, the runHarness will start at the maximum found in the previous iteration, will cut the minRateStep in half, and will set the initialRateStep to twice the new minRateStep. This provides a way to narrow in on the maximum with smaller increments.
  - Default: 0

It is often desirable to be able to generate charts of throughput versus response-time over a range of utilization values. The maximum-finding strategy provides a

convenient way to collect the required data. If you specify a low number of users for the starting run, and a rate step that will give multiple data-points between the initial value and the maximum, the run-script will automatically collect data over a range of loads. A summary of each run will be placed in weathervaneResults.csv.

The target utilization mode works as follows:

1. The run script performs a run at the specified number of users.
2. After the run, the run script calculates the average CPU utilization of the server VMs or the target tier. It uses this value to compute the CPU-utilization per-user. It uses this per-user value to adjust the number of users based on the difference between the measured and target utilizations.
3. The run script then re-runs the benchmark at the new number of users.
4. This continues until the measured utilization is within a given margin of the target utilization.

The command-line parameters for Target utilization mode are:

- **targetUtilizationServiceType**: This controls which service tier the run harness attempts to drive to the target utilization.
    - Possible values: lbServer, webserver, appServer, dbServer, nosqlServer, msgServer, fileserver
    - Default: appServer
- **targetUtilization:** This is the target utilization to be used. The utilization is specified as a whole number value. For example, 50% is specified as 50, and not as 0.5. Note that the target utilization mode requires that logLevel be set to 2 or higher.
    - Default: 70%
- **targetUtilizationMarginPct**: The measured utilization must be within this percentage of the target in order to the run script to declare success. For example, if the target utilization is 50%, the measured utilization must be within +/-(2%*50%), or between 49% and 51%.
    - Default: 0.2

Run-Phase Specific Parameters

The run harness has parameters that control the behavior of specific run phases.

- Power Control:

  The run harness can use the deployment configuration parameters (Section 8.7.1.2) to determine which VMs should be powered on or off for a run. This only works if you have defined the number of VI Hosts and the appropriate DNS mappings, or defined an explicit list of VI Host instances. In the current release, power control is supported only on VMware vSphere.

- **powerOnVms**: If the powerOnVms parameter is set to true, the run-script will attempt to make sure that all required VMs are powered-on on the VI hosts. Power control is currently supported only for VMware vSphere. In order for this to work, passwordless-ssh must be set up for all ESXi hosts.
  - **Default**: false
- **powerOffVms**: If the powerOffVms parameter is set to true, the run-script will attempt to make sure that all VMs on the VI hosts that are not needed for the run are powered off. Be careful when using the parameter to ensure that the run harness will not power off VMs that you wish to remain powered-on.
  - **Default**: false

- Configuration
  - The configuration phase uses the deployment configuration parameters (Section 8.7.1.2) and component tuning parameters (Section 0) when editing the service configuration files. There are no specific parameters for this phase.
- Redeployment
  - **redeploy**: If the redeploy parameter is set to true, the run harness will redeploy all application binaries and web content to the workload drivers, application servers, and web servers. This is mainly used when upgrading to a new version of Weathervane.
    - **Default:** false
- Data preparation
  - **maxUsers**: The maxUsers parameter controls the amount of data to be preloaded in the data services. Enough data will be loaded to support at most this number of users, unless the number of users specified in the run is greater than maxUsers. In that case the run harness load enough data for the specified number of users. It is useful to set maxUsers to the largest number of users that you expect to use in your testing. In that way, the data for the Auction application will only need to be preloaded once.
    - **Default:** 300
  - **maxDuration**: The maxDuration parameter affects the amount of data to be preloaded in the data services. Enough data will be loaded to support a steady-state of at most this duration. However, if the run uses a total duration that is greater than the value of the maxDuration parameter, then the run harness will load enough data for the total duration of the run. It is useful to set maxDuration to the length of the longest run that you expect to use during your testing. In that way, the data will only need to be loaded once, and

you will verify that you have sufficient storage configured for your longest runs. This parameter is specified in seconds.

- **Default**: 7200 (equivalent to 2 hours)

- **loadDb**: The loadDb parameter is used to control whether the run harness will load the data services when trying to perform a run for which the data is not loaded at the proper number of users. When set to false, the harness will exit rather than loading the data.
  - **Default:** true
- **backup**: The backup parameter is used control whether the run harness creates a backup of the data that is loaded into the data services. This only controls whether a backup is created just after the data is loaded. To create a backup of already loaded data, use the rebackup parameter.
  - **Default:** false
- **reloadDb**: The reloadDb parameter is used to force the run harness to load the data services for the specified maxUsers (or users, whichever is larger) for the run even if they are already properly loaded. This parameter is useful when changing the configuration in such a way that the existing data is no longer valid. Such situations include changing the number of MongoDB shards or updating the version of Weathervane.
  - **Default:** false
- **rebackup**: The rebackup parameter is used control whether the run harness creates a new backup of the data that is loaded in the data services, even if a backup already exists. This parameter is useful when changing the configuration in such a way that the existing data is no longer valid. Such situations include changing the number of MongoDB shards or updating the version of Weathervane. rebackup may also be used to force the creation of a backup when the data has been loaded on a previous run.
  - **Default:** false

- Run
  - The run phase is affected by the run length parameters and the logLevel parameter discussed in Section 8.7.1.1.
  - **startStatsScript**: The startStatsScript parameter takes the name of a shell script that should be executed when the run reaches the start of the steady-state. This can be used to start the collection of statistics that are not already collected by the run harness. The script is called with one parameter, which is the length of the steady-state in seconds. The script will run in its own process, and so does not have to return immediately. If the value of the

parameter starts with a /, then it is taken to be the absolute path to the file. Otherwise the location of the script file is assumed to be relative to weathervaneHome.

        - **Default**: The default is no script

    - **stopStatsScript**: The stopStatsScript is the name of a shell script that should be executed when the run reaches the end of the steady-state. This can be used to stop the collection of statistics that are not already collected by the run harness, or to collect up statistic or log files. The script will run in its own process, and so does not have to return immediately. If the value of the parameter starts with a /, then it is taken to be the absolute path to the file. Otherwise the location of the script file is assumed to be relative to weathervaneHome.

        - **Default**: The default is no script

- Data and log gathering
    - The operation of the data and log gathering phase is affected by the logLevel parameter discussed in Section 8.7.1.1.

Component Tuning Parameters

The Weathervane run harness has a number of options that can be used to alter tuning parameters of the various software components involved in the benchmark. This section discusses these options. The options are grouped by software component. It is also possible to apply additional tunings by editing the configuration files for the services that are located under /root/weathervane/configFiles. Note that you should not directly edit the configuration file parameters that are covered by run-harness parameters, as your changes will be overwritten during the configuration phase of a run. You should also not edit the configuration files directly on the service hosts, as those files will also be overwritten during the configuration phase of a run.

Workload Driver Tuning Parameters: These options tune the workload-driver. The key parameter is the JVM heap size, which will need to be increased for large runs. The number of driver threads should also be increased as the size of the driver VM (or native system) is increased.

- **driverJvmOpts**: The command-line options for the workload driver JVM can be set using this parameter. This allows you to alter things like the heap-size, the GC algorithm, etc. The JVM heap size specified by this parameter will need to be increased at higher loads, or the driver will become a bottleneck.
    - Default: -Xmx2G –Xms2G -XX:+AlwaysPreTouch

Application Server Tuning Parameters: These options control some key tuning parameters of the Tomcat instances. The specified tunings are applied to the instances by altering the configuration files of the instances.

- **appServerJvmOpts**: The command-line options for the appServer JVM can be set using this option. This allows you to alter things like the heap-size, the GC algorithm, etc.
    - Default: -Xmx2G –Xms2G -XX:+AlwaysPreTouch
- **appServerThreads**: This controls the number of Java threads available to the Tomcat HTTP executor in each Tomcat instance.
    - Default: 50 per CPU
- **appServerJdbcConnections**: This controls the number of connections available to the Tomcat JDBC connection pool. This is the maximum number of open connections each Tomcat instance can have to the database.
    - Default: 51 per CPU
- **prewarmAppServers:** As discussed in Section 5.2, the run harness will instruct the Configuration Manager to pre-warm the application servers before a run if this parameter is set to true. Pre-warming the app servers allows the use of significantly shorter ramp-up times, but does require the use of a configuration manager in the deployment and does delay run startup time. Pre-warming app servers can be disabled by setting this parameter to false.
    - Default: true

Apache Httpd Tuning Parameters: The run harness provides a number of parameters related to tuning Apache Httpd 2.4. However, the harness is capable of managing the tuning of these parameters, as their proper sizing is related to the number of users and the number of web server nodes. The parameters and their default values are listed in the weathervane.config file.

Nginx Tuning Parameters: The run harness provides a number of parameters related to tuning Nginx. However, the harness is capable of managing the tuning of these parameters, as their proper sizing is related to the number of users and the number of web server nodes. The parameters and their default values are listed in the weathervane.config file.

MySQL Tuning Parameters: The run harness allows for two strategies in tuning the configuration of MySQL. By default, it will automatically adjust the values of some key MySQL configuration variables based on the workload and on the amount of memory configured on the MySQL host. However, it is possible to override the automatic tuning by setting the values of the parameters. The MySQL tuning parameters provided by the harness are:

- **mysqlInnodbBufferPoolSizePct**: This parameter controls the auto-tuning of the MySQL InnoDB buffer pool size. The run harness determines the amount of memory on the MySQL VM, and then sets the buffer pool size to this percent of the total.
    - **Default** = 0.75

- **mysqlInnodbBufferPoolSize**: This parameter explicitly sets the MySQL InnoDB buffer pool size to a particular value. Setting this parameter disables auto-tuning as controlled by the mysqlInnodbBufferPoolSizePct parameter. Set this as you would the value in the my.cnf file. For example 4G corresponds to 4 gigabytes.
- **mysqlMaxConnections**: By default, the run harness automatically tunes the max connections to MySQL based on the number of application server VMs and the size of their database connection pools. Use this parameter to override the auto-tuning and to provide a particular value.

PostgreSQL Tuning Parameters: The run harness allows for two strategies in tuning the configuration of PostgreSQL. By default it will automatically adjust the values of some key PostgreSQL configuration variables based on the workload and on the amount of memory configured on the PostgreSQL VM. However it is possible to override the automatic tuning by setting the values of the parameters. The PostgreSQL tuning parameters provided by the harness are:

- **postgresqlSharedBuffersPct**: This parameter controls the auto-tuning of the size of the PostgreSQL shared buffers. The run harness determines the amount of memory on the PostgreSQL VM, and then sets the shared buffers size to this percent of the total.
    - **Default** = 0.25
- **postgresqlSharedBuffers**: This parameter explicitly sets the PostgreSQL shared-buffers size to a particular value. Setting this parameter disables auto-tuning as controlled by the postgresqlSharedBuffersPct parameter.
- **postgresqlEffectiveCacheSizePct**: This parameter controls the auto-tuning of the PostgreSQL effective cache size. The run harness determines the amount of memory on the PostgreSQL VM, and then sets the effective cache size to this percent of the total.
    - **Default** = 0.65
- **postgresqlEffectiveCacheSize:** This parameter explicitly sets the PostgreSQL effective cache size to a particular value. Setting this parameter disables auto-tuning as controlled by the postgresqlEffectiveCacheSizePct parameter.
- **postgresqlMaxConnections**: By default, the run harness automatically tunes the max connections to PostgreSQL based on the number of application server VMs and the size of their database connection pools. Use this parameter to override the auto-tuning and to provide a particular value.

MongoDB Tuning Parameters: There is only one tuning parameter related to MongoDB.

- **mongodbUseTHP**: For best performance, MongoDB recommends that transparent huge pages be turned off on Linux distributions that support

them. By default, the run harness will disable THP on all MongoDB VMs. If you don't want the harness to disable THP on those nodes, set this parameter to true.

NFS Tuning Parameters: The run harness provides a number of tuning parameters related to NFS. They are:

- **nfsProcessCount**
  - default = 32
- **nfsRsize**
  - Default = 65536
- **nfsWsize**
  - Default = 1048567
- **nfsServerAsync**
  - Default = true
- **nfsClientAsync**
  - Default = true

9 Running the benchmark

9.1 Introduction

The Weathervane benchmark can be used to measure the performance of enterprise infrastructure by driving a workload against the Auction application. The workload emulates simulated users logging in to the application, browsing and joining active auctions, following the progress of the auctions, and bidding on items. A run of the Weathervane benchmark generates the following types of data:

- A Pass/Fail result. See the discussion in Section 6.4.
- Response-Time data for all operations in the workload, including counts of the number of operations that failed their response-time goals.
- Counts of operations and metrics regarding the overall operation mix.

Weathervane can also collect additional information during a run. The amount of information collected is controlled by a parameter in the Weathervane configuration file. The data that can be collected includes:

- Log files from the various software components
- Complete sar data for all Weathervane hosts in the system. This includes CPU, memory, disk, and network performance statistics.
- Java garbage-collection data

- Service-specific performance data from the database, web server, etc.
- esxtop performance data from ESXi hosts

Taken together, this data can be used to understand the effect of different configuration and tuning options on the performance of enterprise applications with characteristics similar to Auction.

The primary interface to Weathervane is the Weathervane run harness, which automates all steps in the execution of the benchmark. The parameters for a run of the benchmark are specified in a configuration file, which is read by the harness at the start of a run. These parameters include the number of simulated users, the length of the run, the number of instances of each type of service, and the amount of data to collect. The Weathervane run-script can also can manage the execution of a series of runs with goals such as finding the maximum user-load for which your deployment can give a passing result, or driving the VMs for a particular service to a target utilization. All of the output of a run is stored on the host on which the run harness is executed, and a summary of the results is placed in a csv file for easy reference and analysis.

9.2 Quick Start

Once you have finished creating your Weathervane deployment, the following steps are all that is necessary to run the Weathervane benchmark:

1. Create a configuration file to prepare the parameters for the run.
2. Run the benchmark using the Weathervane run harness.
3. Look at the results in weathervaneResults.csv and in the output directory /root/weathervane/output/n.

The information contained in the remainder of this chapter is intended to clarify each of these steps.

9.3 Run Phases

Each run of the Weathervane benchmark moves through a series of phases. These are: power control; pre-run cleanup; redeployment; configuration; data preparation; service start; pre-warm application servers; run; service stop; data and log gathering; post-run cleanup; log and data analysis; and results output. A brief description of each phase is given below.

- Power control: In the power-control phase, the run harness powers on VMs that will be needed during the run, and powers off those that will not. This can be convenient when scripting multiple runs with different configurations. Power control is currently only supported on the vSphere VI, and is disabled by default.
- Pre-run cleanup: In this phase, the harness makes sure that all services are shut down, and then cleans up any log or statistics files left over from previous runs.

- Redeployment: In this optional phase, the harness will redeploy all binaries and application artifacts, such as static web content, to the workload drivers, data managers, and application services. This allows in-place upgrading to a new version of Weathervane. The redeployment phase is disabled by default.
- Configuration: The baseline configuration files for each service are stored on the run-harness host under /root/weathervane/configFiles. In the configuration phase, the harness adjusts the configuration files for each service according to Weathervane configuration parameters and then copies them to the proper location on the service hosts. If you want to manually edit a service configuration to adjust performance tunings not handled by the harness, you should make your changes to the files in /root/weathervane/configFiles. Otherwise the harness will overwrite your changes.
- Data preparation: The data preparation phase covers a number of activities related to loading and preparing the data that is used by the Auction application during a run of the benchmark. Due to its complexity, the data preparation phase is covered in more detail in Section 9.4.
- Service Start: In this phase, the harness starts all of the services that will be used in a run.
- Pre-warm application servers: By default, the run harness will pre-warm the application servers so that performance is better even on runs with little or no ramp-up. This assumes the use of a configuration manager. If you don't want the pre-warming, specify "prewarmAppServers" : false, in your configuration file, or --noprewarmAppServers on the command-line. Prewarming does the following:
    - Warm-up the application-level caches.
    - Force loading of most used Java classes
    - Force compiling of code on most common code paths

  While pre-warming adds a few minutes to the start-up of each run, it also allows you to use a much shorter ramp-up period for your runs, leading to an overall decrease in run-time. Note that when using the Elasticity Service, application servers that are added during a run will always go through a pre-warming phase before being integrated into the active configuration. See Section 12 for more information.
- Run: This phase covers the actual execution of the benchmark. The run phase has three sub-phases: ramp-up, steady state, and ramp-down. There are configuration parameters that control the duration of each sub-phase. Performance statistics for the workload and for the services are collected only during the steady state. The harness also provides callouts to external scripts at the start and end of the steady-state to enable additional data collection.

- Service Stop: In this phase, the harness stops all of the services that will be used in a run.
- Data and log gathering: One the services are stopped, the harness gathers up all of the data and log files that were generated during the run and places them in the output directory for the run. The amount of data collected is controlled by the logLevel configuration parameter, as discussed in Section 8.7.1.1.
- Post-run cleanup: Once all files have been collected, the harness deletes them from the service VMs in preparation for the next run.
- Log and data analysis: In this phase, the harness analyzes the run logs and collected data in order to determine whether the run passed, and to prepare summary data for output.
- Results output: The final step in a run is for the harness to print a summary of the run results into a csv file that can be used for reference and analysis. The amount of data that is placed in the results file will depend on the logLevel parameter.

Many of these phases have run parameters that affect the operation of the harness during the phase, or that can cause the harness to skip the phase. The parameters were discussed in Section 8.7.

9.4 Data-Preparation Phase Details

Overview

In order to execute a run of the benchmark, the Weathervane run harness must first ensure that all data services are preloaded with the data that will be needed by the Auction application during the run. The Auction application uses data from a number of sources during each run. These sources are discussed in Section 0. This data must be created and loaded into the appropriate services before a run can be started. Loading the data into the data services can be time consuming, particularly when loading data for large numbers of users. Section 0 provides more information about the amount of storage space needed for the pre-loaded data.

In order to shorten the data preparation phase, the run harness uses two techniques to reduce the need to reload the data before each run. The first is to cleanup and re-prepare already loaded data rather than reloading from scratch between runs. The second is to restore from backups, when possible, if the data is not already properly loaded for the required number of users.

The amount of data to be loaded is controlled by the *users*, *maxUsers,* and *maxDuration* parameters. The Auction application must be preloaded with data to support runs of up to a certain maximum duration with up to a certain number of active users. If insufficient data were loaded, runs would fail due to a lack of available user accounts or an insufficient number of items in the auctions. At the start of the data-preparation phase, the run harness starts the data services and

checks on the number of users and maximum duration that can be supported by the currently loaded data. If the data can support the number of users in the current run, as well as its run duration, then the run harness will clean the data to remove any changes that may have been made on a previous run. It first deletes all data that was added during the previous run. It then resets the state of any records that were updated to their original values. Finally, it compacts all database tables and MongoDB collections to return the storage layout to its original state. These operations can be performed much more quickly than reloading the data services with new data.

If the data services are not loaded for enough users to support the current run, or are loaded for a different configuration, such as for a different image-store implementation, then the run harness can restore the data from a backup rather than loading the data from scratch. This can only be done if a backup was created when the data was originally loaded. Whether the harness creates a backup is controlled by the *backup* configuration parameter. Backups are created in the directories discussed in Section 9.5, and additional disks should be mounted on those directories if backups are to be used. Note that restoring from a backup is not always faster than loading from scratch. You may want to perform tests to determine the optimal strategy for your storage configuration. Backups are not needed when all runs will be less than a known number of users. Because backups require additional storage capacity, you should consider whether they would be useful in your testing environment.

If the current data will not support the run, or if there is no data loaded, and there is no backup available, then the run harness will load the data sources. The number of users for which it loads the data is the maximum of the *users* and *maxUsers* parameters. The run duration for which it loads the data will be the maximum of the total run duration (*rampUp* + *steadyState* + *rampDown*) and *maxDuration*. It is best practice to set maxUsers and maxDuration to be the largest number of users and longest duration you expect to use in your tests. That way the data will only need to be loaded once, and no backup will be needed.

Data Preparation Sub-Phases

The run harness uses the following procedure for the data preparation phase.

1. Check for a reload request

   A reload of the database can be explicitly requested using the *reloadDb* parameter. This may be used when a configuration change makes it necessary to over-write previously loaded data. Note that the run harness will detect many configuration changes and reload automatically. For example the harness can detect a switch from using a NoSQL image-store to a file system-based image store, or changing the number of MongoDB shards. If a reload has been requested, then the run harness will reload the data services. It will then continue with step 3.

2. Check whether the data services are already loaded for the specified number of users and run duration.
    a. If the data is properly loaded, then the harness continues data preparation with step 4.
    b. If the data is not loaded, then the run harness will perform the following steps.
        i. Check whether a backup is available for the current number of users, run duration, and data service configuration. If a backup is available then it will be restored. The harness will then continue with step 3.
        ii. If no backup is available, the harness will load enough data to support the larger of the number of users specified by the maxUsers parameter or the number of users specified for the run, as well as the larger of the total run duration and the maxDuration parameter. The loading of data can be disabled by setting the *loadDb* parameter to false. In that case, the harness will exit rather than loading the data services. You might want to disable loading if you do not want the harness to initiate runs that are outside of a particular value of maxUsers.
3. Check whether the creation of backups is enabled.

    If backup creation is enabled, by setting the *backup* parameter to true, then the harness will create a backup that can be used for future runs with the same maxUsers, maxDuration, and data service configuration.
4. Prepare the data for the current run.

    In this step, which is performed for every run, the harness will ensure that any changes that were made to the loaded data in a previous run are undone, and the data is returned to its original condition. The harness will also edit the auction start-time data to configure the correct number of auctions to be active during the run.

Data Services

There are three services that manage data for the Auction application. These services, and the types of data that they manage are as follows:

- Relational Database: The relational database contains all of the data that is involved in the transactional business processes of the Auction application. This includes data about the past, current, and future auctions and items, the user account data, as well as data used to track the current state of active auctions.
- NoSQL data store: For the Auction application, the NoSQL data store contains two types of data. The first is non-transactional, high-volume, event data about that occur in the Auction application. This is data that is

primarily used for display of historical information. Examples are attendance records that show which users attended which auctions, and data about all bids, whether or not they were the winning bids for an item. The second type of data is the meta-data for images used by the Auction application. The image meta-data describes the image and provides a pointer to the image location.

- Image-Store data: The image-store contains the image files for all images used by the Auction application. Auction provides image-store implementations that support storing the images either in the NoSQL data store, or on a file system. If the images are stored in the NoSQL data store, they will be stored in the same location as all MongoDB data (/mnt/mongoData) on the MongoDB VMs. If the images are stored in a file system, the file system should be mounted on /mnt/imageStore on the file server host.

Data Storage Requirements

The amount of data that will be loaded in the data services is roughly proportional to maximum number of users to be supported by the data. Table 6 shows the approximate amount of disk space required for each data service on a per-user basis, assuming the default maxDuration of 7200 seconds. Note that the exact values will vary depending on the service implementation used. The table also gives an indication of the amount by which the storage usage will increase over the course of a run.

Table 6 Data Storage Requirements

Data Service	Initial storage required per-user (MB) at 7200 sec maxDuration	Additional initial storage for each additional 1000s of maxDuration (MB)	Storage growth per user per 1000 seconds of run-time (MB)
NoSQL Store	0.11		
Image Store	2.50		0.24
Database	0.05		

At smaller numbers of users the values will be slightly larger as the base storage requirements for the data services (e.g. system tables) will become more significant.

These numbers can be used to compute the approximate storage needs as follows. Assume you want to be able to run for 24 hours at a load of 50,000 users. To run for 24 hours, you would need to set maxDuration to 24*60*60 = 86400 seconds. The total storage on the image store would need to be:

2.5 * 50000 + x * ((86400-7200)/1000) + 0.24 * (86400/1000) * 50000

= xex MB

Note that if the images are being stored in the NoSQL data store, then the storage on that tier would need to be the total of the NoSQL store and Image store values.

9.5 File layout

After running the auto-setup script, the Centos 7 host will be configured to run all components of Weathervane. The key files and directories are:

- /root/weathervane : This directory contains the run-script, all of the executables, and the support scripts for the Weathervane benchmark. The results of the Weathervane runs are also stored here in the output subdirectory.
- weathervane.pl: This script is the interface to the run harness used to run the benchmark.
- weathervane.config: This is the default configuration file for the Weathervane run-script. It is read by the Weathervane run-script to control the benchmark runs. All of the configuration-file parameters are documented in this file.
- weathervane_users_guide.pdf: This document.
- runmany.sh: This is a shell script that contains example invocations of the weathervane.pl script. It can be edited and used to run multiple successive invocations of the run script.
- /mnt: This is a standard Linux location for mounting additional volumes. The Weathervane VM has directories under /mnt that are used for database, MongoDB, and image data. In a full deployment, these directories will typically be used as mount points for independent disks.
    - Database Directories: The Weathervane VM is configured to support the PostgreSQL and MySQL databases. The directory structure for the database files makes it easy to mount separate disks for data, logs, and backups.
        - /mnt/dbData: Mount a disk here to add additional space/spindles for database data
            - /mnt/dbData/mysql: The location for the MySQL data files.
            - /mnt/dbData/postgresql: The location for the PostgreSQL data files.
        - /mnt/dbLogs: Mount a disk here to add additional space/spindles for database log files
            - /mnt/dbLogs/mysql: The location for the MySQL log files.

        - /mnt/dbLogs/postgresql: The location for the PostgreSQL log files.
    - /mnt/dbBackups: This directory is used to store backups of database data loaded for different numbers of maxUsers. Storing backups allows switching between maxUser levels without re-generating the data each time. A restore from a backup may be faster than generating the data initially, particularly at large numbers of users. However, the backups will require extra disk space.
- MongoDB Directories
    - /mnt/mongoData: This is the data directory for MongoDB. When images are stored in MongoDB (the default) this can require a large amount of storage.
    - /mnt/mongoBackup: This is the directory in which backups of MongoDB data will be stored when backups are used.
- Filesystem-based image-store directories
    - /mnt/imageStore: When using the filesystem-based image-store, the images for the Auction application will be stored in this directory. To make the images available to the primary workload-driver and all application server nodes, a network file system such as NFS would normally be mounted on this directory.
    - /mnt/imageStoreBackup: This is the directory in which backups of image data will be stored when backups are used for deployments with a filesystem-based image-store.

Some additional files and directories will be created when the benchmark is executed. You can skip this list on first reading of this manual, but it will be useful for reference once you have run the benchmark. The files are:

- weathervaneResults.csv : After each run completes, the run script parses the data files and places a summary of the data in this file in csv format. This includes the run number, and information about the parameters used during the run, the pass/fail status, and the operation response-times. Depending on the logging level, it may also contain Java garbage-collection data from the web container, database statistics, and a summary of the sar data from every component.
- /root/weathervane/output: The output generated by each run of Weathervane is placed in a unique sub-directory of /root/weathervane/output. It may be necessary to mount a separate disk on this directory in order to have enough space for the output when performing many runs. The amount of space required for the output will depend on the setting of the logLevel parameter. The data for each run is stored in a subdirectory named with the run number.

- /root/weathervane/output/n/run.log: For each run, the actual output of the Weathervane workload driver is stored in a file named run.log in the output directory for that run. This is a useful file to look at after to run to look for errors and more detail about the response-times from each of the application instances.
- /root/weathervane/output/n/console.log: The output of the run harness is echoed into this file.
- /root/weathervane/tmpLog: This directory is used to store files during the run. You can follow the progress of a run by following the run.log file in this directory (tail –f tmpLog/run.log).

9.6 Running the Benchmark

Once the configuration file has been created, running the Weathervane benchmark is as simple as invoking the run harness from a command-line.

$ *./weathervane.pl*

The harness handles starting all of the services, preparing the data, and starting the workload generator. When the run completes the results will be analyzed and a message will be printed stating whether the run passed the response-time and operation mix criteria.

To run the benchmark with the configuration file other than the default, /root/weathervane/weathervane.config, you can use the command-line option *configFile*, as in *./weathervane.pl –configFile=/root/weathervane/weathervane.config.large*.

9.7 During the Run

While a run is executing, the run harness will print messages about what is happening as the set-up and execution of the run proceeds.

Once the workload starts runningthe workload driver will write to a log file located in the /root/weathervane/tmpLog directory. If only one workload is running, the file will be called run.log. With multiple workloads, there will be one file for each workload: run-W1.log, run-W2.log, etc.

Each workload will print periodic performance data into its log file. The periodic output interval is 10 seconds, and for each interval the following data is provided:

- The time into steady-state.
- Throughput in operations/sec
- Average response-time for all operations.
- The total number of operations initiated.
- The number of operations that failed due to errors.
- The number of operations that failed their response-time requirements.

- For each individual operation type: The total number of the operation, the number of that operation that failed their response-time requirements, the response-time requirement (in seconds), and the average response-time (in seconds) of the operations that failed to meet their response-time requirements.

This periodic data can be used to follow the progress and performance of a run.

9.8 Benchmark Results

When a run completes, the run harness will indicate whether the run has passed or failed the QoS requirements. For runs with multiple workloads, it will indicate whether each workload has passed or failed.

The data for the operation response-times and operation mix, aggregated across all of the workload drivers, is contained in the run.log file from the primary driver at the end of the run. In addition, a summary of the results will be placed in the file *weathervaneResults.csv* which is created in /root/weathervane. For every run of the benchmark, the run-script places one line in this file with a summary of the run configuration, results, and collected performance statistics.

All of the data collected for a run is placed in the directory /root/weathervane/output/*n*, where *n* is the run number. For the first run, the run number will be 0. This directory has all of the raw data collected during the run, including the workload-driver output; the various application logs; and the statistics output-files. Some files of particular interest are:

- output/0/run.log. This file will contain the workload-driver output for the run. There may be a small number of errors in this log, particularly ConnectionClosedException errors, even on a passing run. A large number of errors will indicate that either some component of your deployment is saturated or that there is some error in your configuration.
- output/0/setupLogs: This directory will contain log files written by the harness during the configuration and start-up portion of a run. You should look in these logs for error messages if the benchmark is not starting or running properly.
- output/0/configuration: This directory will contain a Weathervane configuration file, weathervane.config.asRun, with all of the parameters set as they were for the run. This will reflect the actual values used, with the configuration files values possibly over-ridden by command line parameters. If the logLevel is 1 or higher, this directory will also contain the configuration files for all of the software services.
- output/0/cleanupLogs: This directory will contain log files written by the harness during the clean-up portion of a run

- output/0/logs: If the logLevel is 1 or higher, this directory will contain the log files collected from all of the software services at the end of a run.
- output/0/statistics:
  - Under workloadDriver/*primaryDriverHostname* there will be csv files containing the details of the periodic output and the per-load-interval output, as well as a summary of the results.
  - If the logLevel is 2 or higher, this directory will contain host-level performance data for all of the VMs used in the run. These will be stored in statistics/hosts/*hostname*. For each host the directory will contain the text-converted sar output, as well as the original sar file. A summary of the host-level data will be contained in statistics/hosts/host_stat_summary.csv. This is a useful file to refer to if you have concerns that the usage is not even across hosts.
  - If the logLevel is 3 or higher, this directory will also contain service-specific statistics for many of the services. These will be stored under statistics/*serviceType*/*hostname*.
  - If the logLevel is 4 or higher, it will also contain performance statistics gathered from your virtual infrastructure hosts, such as esxtop data for vSphere ESXi hosts. For esxtop data, the harness will place a parsed summary of the data in the appropriate subdirectory. This file will have a name such as statistics/vsphere/AuctionEsx1/AuctionEsx1_esxtop_report.txt.

10 Running Weathervane using Docker Engine

10.1 Overview

All of the services involved the Auction application, except the filesystem-based image-store, can be run in Docker containers. If you are unfamiliar with Docker, or why you might want to run your application using Docker, you will find documentation and an introduction to Docker at https://www.docker.com.

10.2 Building the Weathervane Docker Images

Overview

In order to run the Auction application in Docker containers, you must first build the Docker images and place them in a Docker registry. Weathervane provides the Dockerfiles needed to build all of the Docker images, along with a script that can be used to build the images and push them to your Docker Hub account, or to a local private registry. Serving the images from a private registry may be necessary when running in a test environment that is isolated from the internet.

For more information about Docker registries, see https://docs.docker.com/registry/.

Weathervane ships with a script to build the Docker images and push them to either a personal Docker Hub account (See https://hub.docker.com/), or to a private registry. You must use one of these methods in order to run the Auction application in Docker containers. The script is called buildDockerImages.pl and is located at /root/weathervane.

Before building the Docker images, you should be sure that the Weathervane executables are up to date. If you have made any code changes, or have pulled in changes from the GitHub repository, then you should first re-build the executables as discussed in Section 3.5.

Building and Pushing to a Docker Hub account

In order to push the Docker images to a personal Docker Hub account. You must first create an account at https://hub.docker.com/. Then run the script from the /root/weathervane directory as follows:

./buildDockerImages.pl –username *username* –password *password*

Where *username* and *password* are the user name and password for your Docker Hub account. This will build the Docker images and push them to your account. Running the script will create a log file called buildDockerImages.log. You should check this log file after running the build script to check for errors.

Building and Pushing to a private repository

In order to push the Docker images to a private registry, you must first configure and start the registry as discussed at https://docs.docker.com/registry/. If you are using an insecure registry, then you will need to add the following to the file /etc/system/system/docker.service.d/docker.conf on every host on which you will run Docker, placing it at the end of the ExecStart line:

--insecure-registry *hostName*:*portNumber*

where hostName is name or IP address of host on which your registry is running, and portNumber is the portNumber on which the registry is listening.

When your registry is running, build and push the images by running the script from the /root/weathervane directory as follows:

./buildDockerImages.pl –host *hostName* –port *portNumber*

This will build the Docker images and push them to your private registry. Running the script will create a log file called buildDockerImages.log. You should check this log file after running the build script to check for errors.

10.3 Using the Docker Images

Running Services in Docker Containers

It is possible to run every service from the Auction application in a Docker container. It is also possible to run only specific service tiers, or even only specific service instances. You indicate which services to run in Docker containers by using the useDocker parameter in the Weathervane configuration file. The useDocker parameter was introduced in Section 8.7.1.3.

If useDocker is set to true at the top-level of the configuration file. Then all services, except the image-store filesystem if being used, will run in Docker containers. There is not currently a Docker image for the filesystem.

If using the explicit configuration format, as discussed in Sections 8.6 and 0, then you can indicate which specific services to run in Docker containers by setting useDocker to true in the JSON object that describes the relevant services.

The Weathervane run harness will handle all configuration of service running in and out of Docker containers. In particular, it will make sure that all configuration files have the correct port numbers for affiliated services, even if the services are running in containers on a bridged network.

In order for the run harness to find the Docker images when setting up a deployment, you must set the dockerNamespace parameter, as discussed in Section 8.7.1.3.

Service-Related Docker Parameters

Weathervane provides parameters that can be used to affect the configuration of the Docker containers for individual services. One of these is used to configure the network for the container, and the others set up the various resource controls provided Docker (https://docs.docker.com/engine/admin/resource_constraints/). While these can be set globally for all containers, they really only make sense when set at the level of individual services.

The parameters are:

- dockerNet: This is used to set the network for the docker container
  - Default: bridge
  - Allowed values: bridge, host
- dockerCpus: Equivalent to –cpus, unless using dockerHostPin. With dockerHostPin==true this specifies how many CPUs to which the container should be pinned.
  - Default: Not set by default
- dockerCpuShares: Equivalent to –cpu-shares
  - Default: Not set by default

- dockerCpuSetCpus: Equivalent to –cpuset-cpus. Do not use this parameter when using dockerHostPin.
    - Default: Not set by default
- dockerCpuSetMems: Equivalent to –cpuset-mems.
    - Default: Not set by default
- dockerMemory: Equivalent to –memory.
    - Default: Not set by default
- dockerMemorySwap: Equivalent to –memory-swap
    - Default: Not set by default

Host-related Docker Parameters

Weathervane provides two parameters that apply to all services running on a Docker host. These parameters are used to control pinning the containers running on the host to particular CPUs using the –cpuset-cpus parameter to the docker run command. These options will rarely be used but may be helpful when investigating performance issues related to CPU scheduling.

- dockerHostPin: When set to true, Weathervane will configure the –cpuset-cpus parameter for all Docker containers so that they are all pinned to a specific set of CPUs. It will assign each container the number of CPUs indicated by that service's dockerCpus parameter. As far as possible, the run harness will pin each container to different CPUs, and will allocate CPUs sequentially. If using this feature, you must set dockerCPUs for each service running in a Docker container, and you should not use dockerCpuSetCpus.
    - Default: False
- dockerHostPinMode: When dockerHostPin is true, this parameter can be used to which CPUs are used when pinning containers. The default is to use all CPUs, but it is also possible to only use even or odd numbered CPUs. This allows using just one thread on cores with hyperthreading enabled.
    - Default: all
    - Allowed values: all, even, odd

11 Weathervane Deployments with Multiple Workloads and Application Instances

11.1 Overview

Up to this point we have focused on deployments of Weathervane which use a single deployment of the Auction application. However, Weathervane supports deployments and runs which use multiple instances of the Auction application. These instances may all be driven by the same set of workload driver nodes, or may be driven by independent workload drivers. Weathervane deployments using multiple application instances are specified hierarchically using two main abstractions: the workload, and the application instance.

A Workload represents a set of workload drivers and a collection of application instances. The client load for all of the application instances is driven by the same set of workload drivers. This means that all of the application instances within a workload must all have the same application type. However, the deployment topologies of the application instances are independent and may be very different. A workload with multiple application instances might be used to scale the load on a group of hosts to a level beyond that which can be achieved by a single application instance. In the current release of Weathervane, a workload is considered to have passed a run only if all of its application instances satisfy the quality of service requirements. See Section 6.4 for more information about QoS requirements.

A Weathervane configuration may have multiple workloads, each with a separate set of workload drivers and one or more application instances. This configuration may also be used to scale to large loads, but has the benefit of separating the QoS requirements of the application instances. When using multiple workloads, the find-max mode of the run harness (see Section 8.7.1.4) can be used to find the maximum of all workloads simultaneously.

An Application Instance represents a collection of services which together comprise a deployment of the Auction application. In the current version of Weathervane, each application instance is a totally independent deployment of the Auction application. Other applications may be supported in future versions of Weathervane. Each application instance also has a Data Manager service which is responsible for loading and preparing the data for that application instance for each run.

The services that make up an application instance are mapped to the hosts on which they should be run. This mapping may be performed using hostname conventions and DNS aliases, or specified explicitly in the configuration file. The data manager is not part of the application deployment, but must also be mapped to a host on which it will be run. For best performance, the data manager is typically mapped to the same host as the database service.

In addition to the workload and application instance, Weathervane uses two additional abstractions that are used when fully describing a deployment. These are hosts and virtual infrastructure. Figure 5 shows the relationships among

those abstractions. As will be discussed in Section 0, these abstractions act as hierarchical containers that allow run parameters to be specified once and applied to all sub-containers without unnecessary repetition.

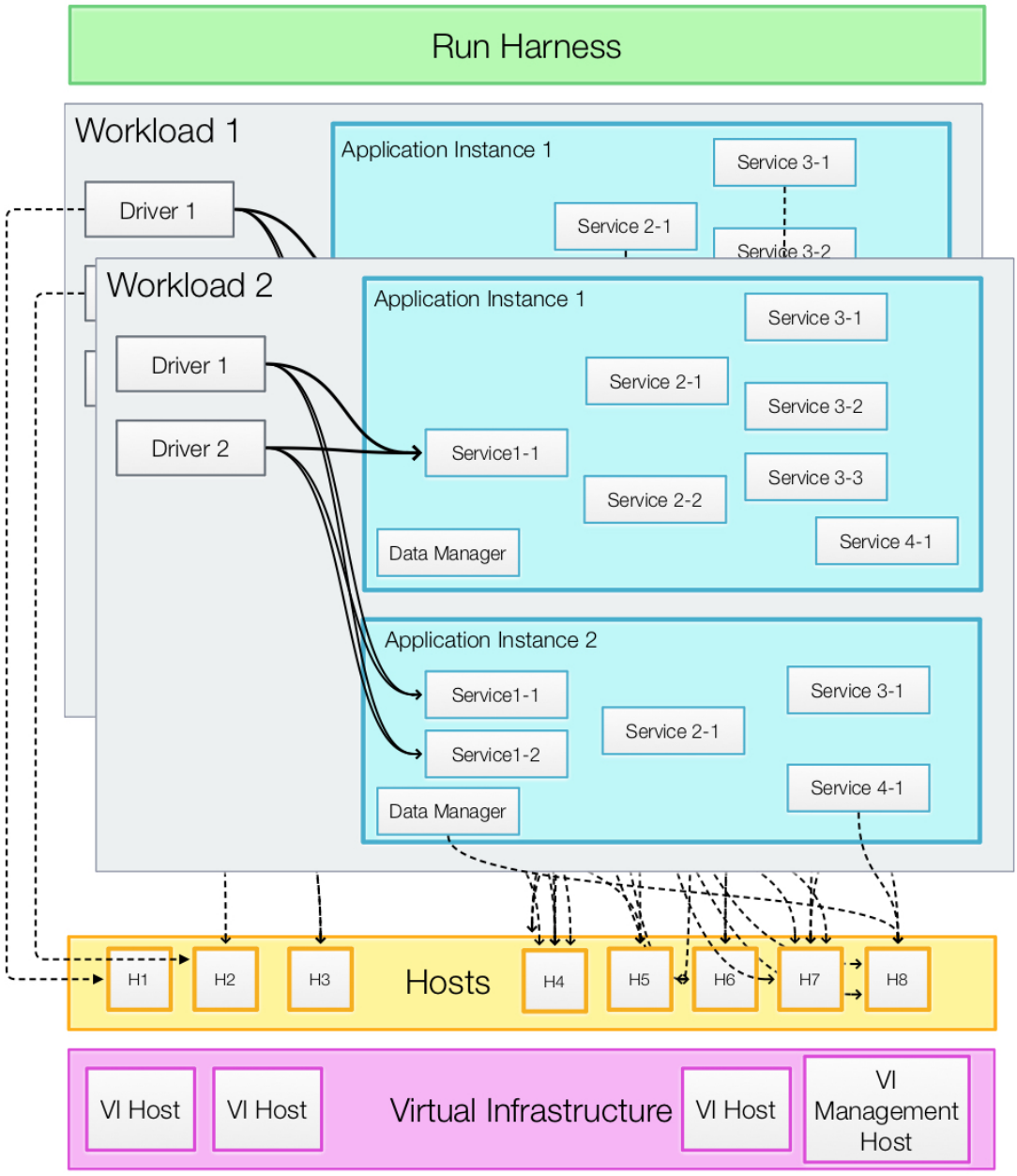


Figure 5 Weathervane Organization

The Hosts abstraction acts as the container for the Weathervane and Docker hosts to be used in a run. Within the Weathervane run harness, the primary identifier for each host is a host name. Host names must be resolvable using DNS. When using a supported virtual-infrastructure, Hosts may also have a VM name, which is an identifier within the virtual infrastructure. The VM name is used

to locate the host within the virtual infrastructure so that the run harness can perform operations on that host's VM.

The Virtual Infrastructure is an optional abstraction that allows you to specify properties of the virtual infrastructure on which your Weathervane hosts are deployed. Specifying the properties of the virtual infrastructure allows the run harness to perform actions such as powering VMs on and off as needed for runs, and collecting performance data from the virtual infrastructure hosts.

11.2 Specifying Multiple Workloads and/or Application Instances with Convention-based Naming

The assignment of hostnames to services based on pre-defined conventions was discussed in Section 8.5. This section describes how the conventions are extended when using multiple workloads or application instances.

When using convention-based naming with multiple workloads and/or multiple application instances, the hostnames for the services include information about the workload and application instance with which they are associated.

For each service, the host name associated with a service is generated using the following pattern, where a '+' denotes string concatenation:

```
Hostname = hostnamePrefix + wkldSuffix + wkldNum
             + appInstanceSuffix + appInstanceNum
             + serviceSuffix + serviceInstanceNum
```

Where each of these components has the following default values:

- **hostnamePrefix**: "Auction"
- **wkldSuffix**: "W"
- **wkldNum**: Numbers from 1 to n, where n is the number of workloads
- **appInstanceSuffix**: "I"
- **appInstanceNum**: Numbers from 1 to m, where m is the number of application instances in a workload.
- ***service*Suffix:** There is a different suffix defined for each type of service supported by the Weathervane run harness. The values are:
  - **workloadDriverSufffix:** "Driver"
  - **dataManagerSuffix:** "Dm"
  - **lbServerSuffix:** "Lb"
  - **webServerSuffix:** "web"
  - **appServerSuffix:** "app"
  - **msgServerSuffix:** "msg"
  - **dbServerSuffix:** "Db"
  - **nosqlServerSuffix:** "Nosql"

- **fileServiceSuffix:** “File”
- **configurationManagerSuffix:** “Cm”
- **coordinationServerSuffix:** “Cs”

- **serviceInstanceNum:** Numbers from 1 to p, where p is the number of instances of a specific service in the workload or application instance.

For example, in a deployment with two web servers in the second application instance of the second workload, the hostnames associated with the web servers would be AuctionW2I2Web1 and AuctionW2I2Web2.

All of the prefix and suffix defaults can be overridden in the Weathervane configuration file. They can be overridden for each application instance, for all application instances in a given workload, or for all workloads.

There are some exceptions to the basic hostname generation rules:

- Since workload drivers exist in a workload, not an application instance, workload driver hostnames do not have the appInstanceSuffix or appInstanceNum. For example, the hostname of the first workload driver associated with workload 2 of a deployment would be AuctionW2Driver1.
- If there is only one workload with only one application instance, the wkldSuffix, wkldNum, appInstanceSuffix, and appInstanceNum are not used. This is the case covered by the simpler conventions discussed in the previously. In this case, for example, the web servers would have hostnames of the form AuctionWeb1, AuctionWeb2, etc. This is done in order to simplify the common case of only one application deployment. If you want to force the harness to assume the full hostname (e.g. AuctionW1I1Web1, etc.) in this case, you can set the parameter “useAllSuffixes” = true in the configuration file. This will simplify the tasks involved in expanding a deployment from a single to multiple application instances.

For each application instance using virtual IP addresses, the www host name is generated using the following pattern, where a ‘+’ denotes string concatenation:

```
wwwHostname = wwwHostnamePrefix + wkldSuffix + wkldNum +
              appInstanceSuffix + appInstanceNum
```

Where **wwwHostnamePrefix** = “www” and the other components have the same definition given previously.

11.3 Hierarchical Configuration Descriptions

Introduction

In order to obtain finer control over the deployment configuration used in a Weathervane run, it is possible to explicitly describe the hierarchy of workloads, application instances, and services to be used, as well as the mapping between

services and hosts. This more complete description style enables a variety of deployment options that are not possible with the basic configuration alone. Some of these are:

- Control over which service instances are to be deployed in Docker containers.
- Application of different tuning parameters to individual service instances.
- Use different implementations for individual service instances. An example of this would be using an application instance with two web servers, one running Nginx and one running Httpd 2.4. This might enable direct comparison of the resource demands of the two implementations under the same load.
- Specify multiple workloads with varying numbers of application instances.
- Specify multiple application instances in a workload with different parameters or services configurations. An example of this might be to have two application instances in a workload, one of which deploys its services in Docker, and another which does not.

These are just a few examples of the flexibility obtained by the using the hierarchical configuration.

The main drawback to the hierarchical specification method is the increased length and complexity of the configuration file. However, the Weathervane run harness provides support for propagating parameter settings to lower hierarchy levels in a way the greatly simplifies the specification of complex topologies.

Full Hierarchical Configuration

A full hierarchical configuration file for a Weathervane deployment explicitly describes the structure of a Weathervane deployment using JSON lists and objects. At the top level, there are three lists of JSON objects, corresponding to the three main structures shown in Figure 5:

- hosts: A list of JSON objects, each of which corresponds to a single Weathervane or Docker host on which services are to be run. A host object must contain a value for the “hostName” parameter. You can also define host-level parameters, such as Docker container pinning, in the host objects.
- workloads: A list of JSON objects, each of which describes the structure of a workload, including workload drivers and application instances.
- viHosts: A list of JSON objects which give the host names of the VI hosts. This is particularly useful when you want to use the host names of VI hosts which are already defined in an existing DNS server.

A workload object contains the following sub-structures:

- A JSON object containing the parameters for the drivers.

- A list of JSON objects describing the application instances.

An application instance object contains the following sub-structures:

- A JSON object containing the parameters for the dataManager.
- A separate list of JSON objects defining the number of instances and parameters for each service type.

A example of a full hierarchical configuration specification is shown below.

```
{
  “hosts” : [
     { “hostName” : “WeathervaneHost1”, },
     { “hostName” : “WeathervaneHost2”, }
  ],
  “workloads” : [
    {
      “drivers” : [ {}, {}, {}, ],
      “appInstances” : [
        {
          “dataManagerInstance” : {
           “hostname” : “WeathervaneHost1”,
          },
          “lbServers” : [ {}, ],
          “webServers” : [{}, {}, ],
          “appServers” : [{}, {}, {}, ],
          “msgServers” : [{}, {}, ],
          “dbServers” : [ {}, ],
          “nosqlServers” : [ {}, ],
        },
        {
          “dataManagerInstance” : {
           “hostname” : “WeathervaneHost2”,
          },
          “webServers” : [{}, ],
          “appServers” : [{}, {}, ],
          “msgServers” : [{}, ],
          “dbServers” : [ {}, ],
          “nosqlServers” : [ {}, ],
        },
      ],
    },
  ],
  “viHosts” : [ {}, {}, {}, ],
}
```

Despite the bare-bones appearance of this structure, this is actually a valid Weathervane configuration file. It defines the structure of a deployment, but accepts the defaults for all parameters, including convention-based naming for the hosts and VMs associated with all of the services. At the top it defines two hosts, WeathervaneHost1 and WeathervaneHost2. The hostname is a required parameter for host instances. It then defines a deployment with one workload, containing three workload drivers and two application instances. The first application instance has a data manager, which will run on the host WeathervaneHost1, one load-balancer, two web servers, three application servers, two message servers, one database, and one NoSQL data store. The load balancer will run on the host AuctionW1I1Lb1, and the hostnames for the rest of the services will also follow the convention-based naming patterns discussed in Section 8.4. The second application instance has a data manager, which will run on the host WeathervaneHost2, one web server, two application servers, one message server, one database, and one NoSQL data store. At the bottom the structure defines a virtual infrastructure with three VI hosts, which will be named AuctionVi1, etc.

In addition to being able to describe structures that cannot be expressed with the basic configuration, this method allows parameters to be over-ridden at the level of individual services. For example, if you wanted to run only the NoSQL server of the second application instance inside a Docker container, you could add the parameter definition *"useDocker" : true,* inside the curly braces defining that instance.

Combined Basic and Hierarchical Configuration

It is possible to combine the basic and hierarchical descriptions so that you add detail only where needed. For example, the previous example, with the assignment of the NoSQL server of application instance 2 to a Docker container, could be written more concisely as:

```
{
  "hosts" : [
      { "hostName" : "WeathervaneHost1", },
      { "hostName" : "WeathervaneHost2", }
  ],
  "workloads" : [
    {
      "numDrivers" : 3,
      "appInstances" : [
        {
          "dataManagerInstance" : {
            "hostname" : "WeathervaneHost1",
          },
          "numLbServers" : 1,
          "numWebServers" : 2,
```

```
            “numAppServers” : 3,
            “numMsgServers” : 2,
            “numDbServers” : 1,
            “numNosqlServers” : 1,
          },
          {
            “dataManagerInstance” : {
              “hostname” : “WeathervaneHost2”,
            },
            “numWebServers” : 1,
            “numAppServers” : 2,
            “numMsgServers” : 1,
            “numDbServers” : 1,
            “nosqlServers” : [ { “useDocker” : true, }, ],
          },
        ],
      },
    ],
  “numViHosts” : 3,
}
```

Parameter Defaults and Hierarchical Overriding

Every Weathervane parameter has either a default value or, for host or VM names, a convention for generating a value. If no value is specified for the parameter, the default or convention is used. It is possible to override default values at every level in the configuration hierarchy. A parameter value defined at an upper level of the hierarchy applies to all relevant objects at lower levels. This makes it possible to override the defaults for many objects or services with one declaration. This value can then be overridden again at a lower level.

For example, consider the following configuration:

```
{
  “workloads” : [
    {
      “numDrivers” : 3,
      “appInstances” : [
        {
          “hostname” : “WeathervaneHost1”,
          “numLbServers” : 1,
          “numWebServers” : 2,
          “numAppServers” : 3,
          “numMsgServers” : 2,
          “numDbServers” : 1,
          “numNosqlServers” : 1,
```

```
            },
            {
              “hostname” : “WeathervaneHost2”,
              “numWebServers” : 1,
              “numAppServers” : 2,
              “numMsgServers” : 1,
              “numDbServers” : 1,
              “nosqlServers” : [
                {
                  “hostname” : “WeathervaneDocker1”,
                  “useDocker” : true,
                },
              },
            ],
          },
        ],
      }
```

In this configuration, the hostName parameter is defined in each appInstance object. This parameter will be applied to all services in that application instance, including the data manager. In the second application instance, the hostName parameter is overridden for the nosqlServer instance. That means that all services for appInstance 1 will run on host WeathervaneHost1, and all services for appInstance 2 will run on WeathervaneHost2, except for the NoSQL service, which will run in a Docker container on host WeathervaneDocker1. Note that we do not have to explicitly define the hosts list, as the run harness will internally create a host object for any hostname definition it encounters. The hosts list is really only needed when defining other parameters for individual hosts, such as Docker container pinning. The host names specified in this way must be translatable to IP addresses by the DNS server.

11.4 Using findMax and targetUtilization modes with Multiple Workloads and AppInstances

The findMax and targetUtilization modes were introduced in Section 8.7.1.4. These modes run the benchmark repeatedly until reaching a user load that is the maximum passing or that drives a service tier to a given CPU utilization. These modes also work when running with multiple workloads and application instances. In both modes, the runs will repeat at different user loads until the maximum or target utilization has been reached for all application instances in all workloads.

12 Variable Loads

12.1 Overview

Up to this point, we have assumed that the user load driven against an application instance is constant for the duration of the steady-state. However, Weathervane allows for the specification and generation of loads that vary over the course of a run. This capability is useful when comparing infrastructure components whose performance may be affected by spikes and drops in load.

Variable loads are supported through the specification of a user load path. A user load path specifies a number of intervals of fixed duration. Within each interval, the number of users is either fixed, or ramps up or down between a start and end point.

12.2 Using Variable Loads

Variable loads are specified by including a user load path in the configuration file. A user load path is a list of intervals. Each interval specifies a duration, and either a fixed number of users to run in that interval, or a start and end number of users for the interval. An example user load path as specified in the Weathervane configuration file is as follows:

```
"userLoadPath" : [
    {"duration" : 300,  "users" : 2000, },
    {"duration" : 120, "startUsers" : 2000, "endUsers" : 4000, "timeStep" : 10},
    {"duration" : 240,  "endUsers" : 1000, "timeStep" : 10, },
    {"duration" : 120,  "endUsers" : 4000, },
    {"duration" : 300,  "users" : 2000, },
],
```

In this example, the first interval lasts 300 seconds. In that interval, the workload driver will run 2000 simulated users. In the second interval, which lasts 120 seconds, the workload driver will start at 2000 users and ramp the load up to 4000 users, with the number of users incremented every 10 seconds. In this case, the increment will be (4000 - 2000) / (120 / 10) = 166.667. Note that the driver will adjust this number so that some intervals increase by 166 and some by 167 so that the total increase is 2000 users.

The third interval demonstrates that the "startUsers" parameter is optional. If omitted, the interval will start with the same number of users as were active at the end of the previous interval. If "startUsers" is omitted from the first interval, then the interval will start with 0 users. This particular example will ramp down from 4000 to 1000 users over 120 seconds.

The fourth interval shows that the "timeStep" parameter is optional. The default timeStep for an interval is 15 seconds.

The final interval will run 2000 users for a fixed duration. In this case, you should note that there is a step-wise change in the number of users from the previous interval, which ended with 4000 users, to this one, which runs 2000 users. This is allowed.

If a userLoadPath is specified, then it overrides any specification of the "users" parameter, even if "users" is specified on the command-line.

Note that the userLoadPath is typically specified per-application-instance. If you are running multiple application instances, each with their own userLoadPath, then the userLoadPaths should be specified within the appInstances block. If you are only running one appInstance then you can specify the userLoadPath at the top level of the configuration

There is also a boolean parameter *repeatUserLoadPath*, which, if true, causes the run harness to repeat the user load changes described by the configuration path when the run length is longer than the total duration of the path. The default value is false, which means that when the last entry of the userLoadPath is reached, the number of users remains the same for the duration of the run. Specify this in the configuration file as: *"repeatUserLoadPath" : true,*. This parameter can be over-ridden at the workload level.

Note that the userLoadPath is independent of the run-duration intervals. The userLoadPath will begin at the start of the run. If the total duration specified in the userLoadPath is less than the total run duration, then the number of users will stay at the last value specified in the userLoadPath for the remainder of the run, unless the repeatUserLoadPath parameter is set top true. If the total duration specified in the userLoadPath is greater than the total run duration, then the periods after the end of the run will not be used.

12.3 Pass/Fail Criteria with Variable Loads

When the load varies over the course of a run, the proportion of operations in the overall mix will vary from that given in Section 6.4. This is unavoidable, as the percentage of login and logout operations will be affected by the load changes, and the rate of bidding on items will be affected by the varying number of users attending each auction. As a result, the pass/fail decision for an application-instance which uses a userLoadPath will not look at the operation mix percentages, but will be based solely on the operation response-times.

When using variable loads, it is also possible that whether the response-times satisfy the 99th-percentile requirement over the course of an entire run many not be the most relevant way to assess performance. It may be useful to inspect the response-time behavior over the separate load intervals. At the end of a run which uses a userLoadPath, the run harness will place a summary of the per-load-interval statistics in a text file under /root/weathervane/output/*n*/statistics/workloadDriver/*driver1hostname*. The file will be called something like appInstance1-loadPath1-*targetIp*-summary.txt. There are also csv files in this directory with the same data in csv form.

13 Application Elasticity in Weathervane

13.1 Overview

In the context of distributed applications, elasticity refers to the capability of an application to scale its use of resources up and down in response to changing loads. The Weathervane Auction application supports elasticity by scaling up and down the number of instances in a service tier. Currently the only tiers that support elasticity are the web server and application server tiers.

The elasticity support in Weathervane is implemented by the combination of the Elasticity Service and the Configuration Manager depicted in Figure 1. The Elasticity Service is the service that is responsible for deciding when service instances should be added or removed. While a sophisticated Elasticity Service might monitor performance metrics to make those decisions, the default Elasticity Service currently provided by Weathervane follows a pre-set schedule to add and remove services over time. This service is discussed in Section 13.2. When the Elasticity Service decides that a service should be added or removed, it sends a message to the Configuration Manager. The Configuration Manager keeps track of the current configuration and can take all necessary steps to add or remove services to the running configuration. Note that the hosts on which the services are to run must be powered on and available before requesting new services from the Configuration Manager.

13.2 Default Elasticity Service

The default Elasticity Service provided by Weathervane allows the application configuration to be controlled during a run by specifying a configuration path. The configuration path specifies a number of intervals of fixed duration, and the number of each service-type to be active in that interval. Currently Weathervane supports changes only in the number of application servers and web servers.

- In the current version of Weathervane, a configuration interval cannot specify a number of application servers that is lower than the number specified in the initial interval. This is due to limitations in the Auction application that will eventually be fixed.

Note that reconfiguration operations can take between 1 and 4 minutes, depending on the change being made, and only one reconfiguration operation can occur at a time. As a result, you should not specify the configuration-path interval durations to be less than 5 minutes (300 seconds), although some operations take much less time. When in doubt, experiment.

Note that both the user load path and the configuration path are independent of the run-length. If a run is shorter than the total duration specified in the path intervals, the intervals at the end of the run are ignored. If a run is longer than the total duration of the path intervals, then the load and/or configuration remains in the final state.

In order to use application elasticity with Weathervane, there are two pieces that must be added to your configuration.

1. You must include a Configuration Manager in your configuration. The Configuration Manager is the service that keeps track of the configuration of the application and executes requests to change the configuration.
    - This is done either by specifying "numConfigurationManagers" : 1, or "configurationManagers" : [{ },], in your configuration file. With the second usage, you can specify parameters such as hostName within the configuration-manager object (the {},).
    - You must also specify a mapping from the configuration manager hostname, which defaults to AuctionCm1, to its IP address in your DNS server or hosts file. Note that the configuration manager uses very few resources and can be run on the same host as another service.
2. You must include an Elasticity Service in your configuration. The Elasticity Service contains the intelligence to decide when to change the configuration. When it decides a change is required, it sends a message to the Configuration Manager to execute the change. There is currently only a single implementation of the Elasticity Service, which requests changes to the configuration based on a user specified configuration path. Future implementations might monitor response-times or resource usage in order to decide when changes are needed, and might interact with the infrastructure to provision or decommission resources.
    - Including an Elasticity Service is done either by specifying "numElasticityServices" : 1, or "elasticityServers" : [{ },], in your configuration file.
    - There is no need to specify a hostname or IP mapping for the current Elasticity Service implementation. It runs completely within the run harness.

A configuration path is specified with the "configPath" parameter in the configuration file. A configPath is a list of intervals, each with a fixed duration. Within each interval, you specify the number of instances of each type of service that should be running during that interval. Note that you can specify numbers for all of the service-types, but only specifications for appServers and webServers will be honored during the configuration changes. Changes in other services might be supported in future releases.

There are a few restrictions with the elasticity support:

- You cannot run MongoDB in sharded mode when adding app servers, only as a single node or replicated.
- The configuration manager does not yet support adding or removing services running in Docker containers.

An example configPath is as follows:

```
"configPath" : [
  {"duration" : 600, "numAppServers" : 2, "numWebServers" : 1},
  {"duration" : 600, "numAppServers" : 3,},
  {"duration" : 600, "numWebServers" : 2,},
```

{"duration" : 600, "numWebServers" : 1,},

{"duration" : 600, "numAppServers" : 2,},

{"duration" : 600, "numAppServers" : 4, "numWebServers" : 2},

],

This configPath consists of six intervals, each of which lasts for 600 seconds. In the initial interval, there are 2 app servers and 1 web server active. In the second interval one additional app server is added. Note that it is not necessary to specify the number of services when the number is unchanged from the previous interval. If the number of a service isn't specified in the first interval, then all of the services specified by the "numAppServers" and "numWebServers" (or "appServers" : [], and "webServers" : [],) parameters are used.

The third interval adds a web server, the fourth removes a web server, the fifth removes an app server, and finally the six adds two app servers and one web server simultaneously. Any combination of services adds and removes is allowed.

Important: The specification of a configPath is in addition to the specification of the app servers or web servers using either the "numAppServer" and "numWebServers" parameters or the "appServers" : [], and "webServers" : [], parameters. For the current implementation of the Elasticity Service, all of the services to be used during a run must be powered on and accessible throughout the run. The number of services specified using these parameters must be greater than or equal to the maximum number used in any configuration interval.

There is also a Boolean parameter repeatConfigPath, which, if true, causes the run harness to repeat the configuration changes described by the configuration path when the run length is longer than the total duration of the config path. The default value is false, which means that when the last entry of the configPath is reached, the application configuration remains the same for the duration of the run. Specify this in the configuration file as: "repeatConfigPath" : true,. This parameter can be over-ridden at the appInstance level.

13.3 Configuration Manager API

13.3.1 Overview

The default Elasticity Service is useful in some limited contexts, but a full implementation of an elasticity service would interface with a number of APIs or data sources:

- A source of performance metrics, both from the application and the underlying infrastructure.
- An API for the infrastructure that allows new VMs or other forms of capacity to be allocated and powered on and off.
- The REST API for Configuration Manager in order to cause services to be added or removed from the application deployment.

This section documents the REST API for the Configuration Manager. This information will be needed when implementing an elasticity service for the Auction application.

The Configuration Manager API supports operations for initialization, configuration queries, and configuration-change requests.

13.3.2 Initialization Operations

The initialization API for the Configuration Manager (CM) is used for three purposes:

- To provide the CM with the defaults to use for the various tuning and configuration parameters related to services that may be added during a run.
- To provide the CM with knowledge of the initial configuration of the application.
- To request the CM to pre-warm an application server before it is brought into service.

The initialization API is used by the Weathervane run-harness to initialize the CM at the start of a run. It will seldom be necessary for an elasticity controller to use this API.

13.3.2.1 Default Setting Operations

The CM API supports operations that set defaults for each of the following services: appInstance, appServer, configurationManager, coordinationServer, dbServer, fileServer, lbServer, msgServer, nosqlServer, webServer. The operations are HTTP POST requests to the following url:

http://auctioncm1:8888/*serviceType*/defaults

Where serviceType would be replaced with one of appInstance, appServer, etc.

The request body for these requests is a JSON object which contains default values for the Weathervane parameters that apply to that service type. For example, the operation to set the app server defaults would look like:

http POST http://auctioncm1:8888/appServer/defaults

with a body that looks like:

{

“appServerJvmOpts” : “-Xmx4G –Xmx4G –XX:+AlwaysPreTouch”,

“appServerThreads” : 50,

…

}

These requests should receive an HTTP 200 response. If not, the body of the response will be a JSON object with a “message” field containing error information.

13.3.2.2 Initial Configuration Operations

The CM API provides operations that allow it to be notified of the initial configuration of the application deployment, including for each of the following services: appInstance, appServer, configurationManager, coordinationServer, dbServer, fileServer, lbServer, msgServer, nosqlServer, webServer. The operations are HTTP POST requests to the following url:

http://auctioncm1:8888/*serviceType*/add

Where serviceType would be replaced with one of appInstance, appServer, etc.

The request body for these requests is a JSON object which contains the information describing the service instance. This will always include the hostName for the service, and may include the values of any Weathervane parameters that override defaults for that service instance. In addition, the JSON object will have a “class” field that contains the service-type and is used by the CM when de-serializing the request. For example, the operation to add an appServer instance to the initial configuration would look like:

http POST http://auctioncm1:8888/appServer/add

with a body that looks like:

{

“class” : “appServer”,

“hostName” : “AuctionApp1”,

}

These requests do not cause the CM to configure or reconfigure any services. They only provide information that is used by the CM when handling configuration change requests.

These requests should receive an HTTP 200 response. The body of the response will be a JSON object that contains an “entityId” field, which has the ID by which information about the service can be accessed, as well as a number of links for performing specific actions against the entities. For example, the response to the above request will have the body:

```
{
  "_links" : {
    "remove" : {
      "href" : "http://AuctionCm1:8888/appServer/1"
    },
    "warm" : {
```

```
      "href" : "http://AuctionCm1:8888/appServer/1/warm"
    },
    "configure" : {
      "href" : "http://AuctionCm1:8888/appServer/1/configure"
    },
    "self" : {
      "href" : "http://AuctionCm1:8888/appServer/1"
    },
    "start" : {
      "href" : "http://AuctionCm1:8888/appServer/1/start"
    }
  },
  "status" : "SUCCESS",
  "entityId" : 1,
  "message" : "Service added to configuration"
}
```


These URLs can be used to interact with the service entity. For example an HTTP GET to http://AuctionCm1:8888/appServer/1 would be used to retrieve a JSON object with all of the information about the service, while an HTTP DELETE to the same URL would remove the service from the configuration.

13.3.2.3 App Server Pre-warm Requests

The CM has the ability to pre-warm the application servers before they are integrated into a deployment of the Auction application. The app servers are pre-warmed by presenting them with a series of requests that have the same profile as those typically issued by a group of Users. These requests initiate class-loading and just-in-time compiling by the app server JVM. They also cause the application-level caches to be pre-loaded with relevant data. By pre-warming the application servers, Users will not experience excess delays when the app server is first integrated into the running configuration. Without pre-warming, the first burst of user requests to the app server will experience longer latencies.

By default, the CM warms all app servers when they are added to a running configuration through a configuration-change operation (see Section 13.3.4). It also provides an API to request that app servers be pre-warmed. This is used by the Weathervane run-harness at the beginning of a run to pre-warm the app servers. Doing this at the start of a run shortens the time required for the ramp-up.

An individual app server can be pre-warmed by doing an HTTP PUT to the warm URL returned when it is added to the configuration, e.g.: http://AuctionCm1:8888/appServer/1/warm.

Multiple app servers can be warmed simultaneously by sending an HTTP PUT to http://AuctionCm1:8888/appServer/warm, where the body of the request is a JSON array with the entityIds of the app servers to be warmed, e.g. “[1, 2, 3, 4]”.

These requests should receive HTTP 200 responses. Note that it can take from 2 to 5 minutes or more to pre-warm, depending on how many app serves are involved.

13.3.3 Query Operations

For each service in a configuration, it is possible to query the values of all parameters for the service by sending an HTTP GET operation to http://auctioncm1:8888/*serviceType*/*entityId*, where serviceType is replaced by one of appInstance, appServer, etc., and entityId is replaced by the integer entityId returned when the service is added to the configuration.

It is also possible to query for the entire configuration at any point by sending an HTTP GET to http://auctioncm1:8888/configuration. The response to this request will be an HTTP 200 response, with a response body of the following form:

```
{
“configurationManagers” : [{“class” : “configurationManager”, “
                                    id” : 1,
                                    “hostname” : “AuctionCm1”
                                    … }],
“lbServers” : [[{“class” : “lbServer”, “
                                    id” : 1,
                                    “hostname” : “AuctionLb1”
                                    … },
                {“class” : “lbServer”, “
                                    id” : 2,
                                    “hostname” : “AuctionLb2”
                                    … },
                ],
…
```

For each service type, this response has a list of JSON objects, each of which has all of the information that describes the service, including the id and the hostname.

13.3.4 Configuration Change Operations

The CM can perform the following changes to an active configuration of the Auction application:

- Remove Web Servers
- Remove App Servers
- Add Web Servers
- Add App Servers

When removing services, the hosts on which the services are running must stay up until the configuration change operation completes. When adding services, the hosts on which the services to run must be powered on and reachable before initiating a configuration change operation.

The CM supports a single operation to perform all of these configuration changes. This allows multiple changes to be performed in one operation. For example, app servers can be added and web servers removed in the same operation. The CM will perform a carefully orchestrated series of configuration changes, including reconfiguring load balancers and reverse-proxy configuration in the web servers, as well as pre-warming any added app servers, in order to make the configuration changes without disrupting outstanding operations.

The CM can handle only one configuration change operation at a time. Once one configuration change request is received, all subsequent requests will block until the previous request finished.

Configuration change requests can take between one to five minutes, or even longer if many app servers are being added at once. You should be sure to use an appropriate timeout value for your HTTP client. The longest change duration will be when adding app servers due to the pre-warming process. It is possible to specify that an app server should not be pre-warmed when added, but this risks performance issues with the initial requests issued to the non-warmed app server.

The configuration change request is an HTTP PUT operation to the URL http://auctioncm1:8888/configuration. The request body is a JSON object with the following format:

{

“appServersToAdd” : [],

“webServersToAdd” : [],

“numAppServersToRemove” : 0,

“numWebServersToRemove” : 0,

}

This particular request body would not make any changes to the configuration.

The response to a configuration-change request should be an HTTP 200 response, with a response-body that look like the following:

{

“appServersRemoved” : [*...List of appServer objects…*],

“webServersRemoved” : [*… List of webServer objects…*],

“addedAppServerIds” : [*…List of appServer IDs…*].

“addedWebServerIds” : [*…List of webServer IDs…*],

“status” : “SUCCESS”,

“message” : “Configuration changed successfully”

}

In order to add app servers or web servers, the JSON arrays for the appServersToAdd or webServersToAdd parameters should be populated with JSON objects describing the parameters for the service. The only parameter that must be included is the “hostName” parameter that tells the CM which host should be used to start the service. The CM will use the defaults populated by the run-harness for any parameter not explicitly set in the configuration change request. The JSON objects must also include a “class” field whose value is “appServer” or “webServer” as appropriate. The “class” field is required for the CM to deserialize the objects correctly. It is also possible to include values for any Weathervane parameter that is valid for the service. For example, you can prevent pre-warming of an app server by including *“prewarmAppServers” : false* in the JSON object for the app server.

For example, a request to add two app servers and one web server would look like:

{

“appServersToAdd” : [{“class” : “appServer”, “hostName” : “AuctionApp9”},

{“class” : “appServer”, “hostName” : “AuctionApp10”}],

“webServersToAdd” : [{“class” : “webServer”, “hostName” : “AuctionWeb5”}],

“numAppServersToRemove” : 0,

“numWebServersToRemove” : 0

}

The response to this request would look like:

{

“appServersRemoved” : [],

“webServersRemoved” : [],

“addedAppServerIds” : [10, 11].

“addedWebServerIds” : [5]

“status” : “SUCCESS”,

“message” : “Configuration changed successfully”

}

The information that is returned for the services that are added are the IDs that have been assigned to the services and that can be used for querying the service information.

When removing services, the configuration change request specifies on how many services should be removed. The CM decides which service instances to remove. This allows it to make intelligent removal decisions based on the role being played by specific service instances.

A request to remove two app servers and a web server would look like:

{

“appServersToAdd” : [],

“webServersToAdd” : [],

“numAppServersToRemove” : 2,

“numWebServersToRemove” : 1,

}

The response to this request would look like:

{

“appServersRemoved” : [{“class” : “appServer”, “hostName” : “AuctionApp2”, … },
{“class” : “appServer”, “hostName” : “AuctionApp3”, …}],

“webServersRemoved” : [{“class” : “webServer”, “hostName” : “AuctionWeb1”, …}],

“addedAppServerIds” : [].

“addedWebServerIds” : []

“status” : “SUCCESS”,

“message” : “Configuration changed successfully”

}

For the services that are removed, the JSON arrays in the appServersRemoved and webServersRemoved fields will contain JSON objects that contain all of the parameters for the service instances that were removed from the running configuration. The most important of these will be the “hostName” field. This will identify the host on which the service was running. If no other services are running on that host then it can be powered off once the configuration-change request is completed.

14 Troubleshooting

15 Performance Tuning for Weathervane

15.1 Overview

The performance that can be achieved by a deployment of Weathervane will depend on the configuration and tuning choices made at each service and infrastructure level. Performance can be affected by the tuning of the application services and operating system; the configuration of the VMs; the capabilities of the virtual-infrastructure; the performance of the physical servers, storage, and networks; and many other factors. In this chapter we give an overview of some of the performance tunables that are most likely to influence the overall benchmark performance. This is not intended to provide a comprehensive introduction to performance tuning, and you will likely need to refer to other resources to achieve the best possible performance from your deployment.

15.2 Operation/Service Interaction

The task of handling each operation issued to the Auction application is shared by many of the services that make up a full deployment of the application. This section gives an overview of which services are involved in each operation. This information may be useful when isolating response-time issue that seem to be affecting only a subset of the operations.

There are a few services that, if present in a deployment, are involved in every operation. These are:

- The load balancer: The load balancer routes each operation to the appropriate web server. It balances the TCP connection load among the web servers, and maintains affinity between clients and web servers based on TLS session information.
- The web server: The web server performs many functions in an Auction deployment:
    - It terminates TLS connections between the clients and the application.
    - It serves static web-page content (html, css, and JavaScript scripts) to the clients. The web interface for the Auction application is a single-page application, and so all of this content is served for the HomePage operation.
    - It acts as reverse-proxy load balancer, spreading requests among the application servers.
    - It acts as a reverse-proxy cache, caching any cachable HTTP responses from the application servers. For the Auction application, cacheable content is limited to images returned by an

application server when the images are stored in the NoSQL data store.

- When the images for the Auction application are stored in NFS, it serves the images directly without forwarding the requests to the application servers.

The application server is involved in every operation that is not handled completely by the web server.

The following table shows which services are involved in each operation. Note that some of the operations involve multiple HTTP requests, each of which may use different services. In that case each HTTP request is noted separately in the following table.

Table 7 Services used by Auction Operations

Operation	Services Used
HomePage	
Login	Application Server Database
GetActiveAuctions	Application Server Database (if application-cache miss)
GetAuctionDetail GetAuctionInfo Request	Application Server Database (if application-cache miss)
GetItemsForAuction request	Application Server Database (if application-cache miss)
GetItemThumbnails requests (up to ten simultaneous requests)	Filesystem (if images stored in NFS and deployment includes a web serve Application Server (if images not in NFS or if web server cache miss). MongoDB (if images in NoSQL service and if application-cache miss)
GetUserProfile	Application Server Database
UpdateUserProfile	Application Server

	Database
JoinAuction	
JoinAuction Request	Application Server MongoDB
GetAuctionInfo Request	Application Server Database (if application-cache miss)
GetCurrentItem Request	Application Server
GetCurrentBid Request	Application Server
GetItemThumbnail Request	Filesystem (if images stored in NFS and deployment includes a web serve Application Server (if images not in NFS or if web server cache miss). MongoDB (if images in NoSQL service and if application-cache miss)
GetCurrentItem	
GetCurrentItem Request	Application Server
GetCurrentBid Request	Application Server
GetItemThumbnail Request	Filesystem (if images stored in NFS and deployment includes a web serve Application Server (if images not in NFS or if web server cache miss). MongoDB (if images in NoSQL service and if application-cache miss)
GetNextBid	Application Server
PlaceBid	Application Server RabbitMQ
LeaveAuction	Application Server MongoDB
GetBidHistory	Application Server

	MongoDB
GetAttendanceHistory	Application Server MongoDB
GetPurchaseHistory GetPurchaseHistory Request	Application Server Database MongoDB
GetItemThumbnails Request (up to ten simultaneous requests)	Filesystem (if images stored in NFS and deployment includes a web serve Application Server (if images not in NFS or if web server cache miss). MongoDB (if images in NoSQL service and if application-cache miss)
GetItemDetail GetItemInfo Request	Application Server Database
GetImagesForItem Request (up to four simultaneous requests)	Filesystem (if images stored in NFS and deployment includes a web serve Application Server (if images not in NFS or if web server cache miss). MongoDB (if images in NoSQL service and if application-cache miss)
GetImageForItem	Filesystem (if images stored in NFS and deployment includes a web serve Application Server (if images not in NFS or if web server cache miss). MongoDB (if images in NoSQL service and if application-cache miss)
AddItem	Application Server Database
AddImageForItem	Application Server MongoDB Image Store
Logout	Application Server Database

16 Change Log

16.1 Overview

This section describes the major changes made in previous releases.