# TomcatGridDiscussion

## Apache Tomcat Grid

This write up is a draft that includes a set of ideas that could be useful when managing multiple Tomcat instances. Even if some of these ideas go beyond the effort we want to undertake, I decided to include them so they can be discussed and maybe included on the road map. This description is by no means comprehensive and/or finished but provides a base line to start bouncing ideas.

Please feel free to update, fix, change, and reorganize this document in any manner you feel is useful.

### Introduction

The Tomcat Grid manages one or more local or remote Tomcat instances from a centralized location. This manager application shows the status of each Tomcat instance, and provides a simple interface to trigger operations on them, individually or as a group.

### Architectural Overview

The centralized location of the Tomcat Grid (the primary location) stores the grid configuration, probably as an XML file. For reliability purposes, secondary locations can be setup so the grid configuration is replicated after every change.

The primary location runs a Tomcat Web Grid Manager, a web application that runs on a separate Tomcat instance. This manager application shows the state of the all Tomcat instances on the grid, and allows users to trigger operations on these Tomcat instances.

In addition to the Web Grid Manager, a Command-line (CLI) Grid Manager application mimics the functionality of the web manager, with an equivalent (text-only) interface that shows the status of the Tomcat instances, and provides commands to trigger remote operations on them.

The secondary locations run on dedicated Tomcat instances. They run a copy of the Web Grid Manager but they are shut down all the time unless they are explicitly started by an operator (when the primary one goes bad). No two managers should be active at the same time. If activated, the secondary locations can start producing changes to the local grid configuration, and these changes are replicated to all other managers (if/when possible).

The example shown in the figure below assumes Machines #1 and #4 run a lighter web and/or back end applications, so they have enough resources bandwidth to run the extra Tomcat instance for the manager. In case the primary manager on Machine #1 goes down (or the whole Machine #1 goes down), there is a secondary manager (on Machine #4) that can be activated to take over the managing duties.

TomcatGridExample.png!

The Grid Agents are Java processes installed and running on each machine that listen (on a configured port) for commands from a Grid Manager application (Web or CLI) and act upon them by interacting with the local Tomcat instances. No encryption is envisioned on the network channels, since all these machines are considered to be installed on a secured network segment (at least in prod). [Maybe we should reconsider this]

*markt: I think this needs to be reconsidered.*

*vlad: Sure. My take was to use unencrypted connections, since it's fast to implement for me and I don't know well how to implement a secured connection : / . Maybe we could even offer both options. Anyway, why do you consider we need an secured connection? In my experience, I've never needed them in dev/state/QA/test, and in prod the environment are isolated. Well... I guess there are other scenarios I haven't been exposed to.*

All Tomcat instances (including the Web Grid Managers), as well as the Grid Agents, are installed manually. The primary grid manager and all the agents are started manually too.

Once the Web Grid Manager is started up, machines and instances can be registered in it, so they can become manageable. No centralized (comprehensive) provisioning is envisioned until later versions of Tomcat Grid (see below).

The Grid Agents are processes that can also be managed. In particular, status can be obtained (and showed) from them, and basic operations (start/stop /kill) can be triggered on them. These operations are, however, heavily dependent on OS and OS capabilities (configuration, installed tools, etc.) and the infrastructure architecture (fire-walled machines, network VLans, etc).

Collectively, Tomcat instances and Grid Agents are "services" since both can be managed.

*markt: Managing the agents strikes me as making this significantly more complex. Operating systems have tools to ensure particular services are running and are restarted if they fail. What is the benefit of pulling this into this tool?*

*vlad: I agree that agents can run as daemons and be up all the time. However, I've seen many "robust" programs to have memory leaks and that for one reason or another stop working normally after some time. Maybe after a couple of days, or weeks, or months. Maybe it would be useful to restart them just before a critical operation such as a deployment. I agree that managing agent is more expensive since it requires the development and maintenance of multiple OS-dependent implementations.*

Later versions of the Grid include "collection" management. This allows to group subsets of services (Tomcat instances and Grid Agents), so they can be operated as whole. Each collection can include plain services, or other collections (recursively).

Below is a general overview of the software modules and their responsibilities.

TomcatGridSoftwareModules.png!

_The modules are:

- **Core module**: Shared logic to be used by all other modules.
  - Core data types, such as "Machine", "Instance".
  - Share core logic. For example, grid configuration file parsing/update.
  - Defines the Grid Agent functions (as interfaces), but does not implement them. These are used by all Managers modules.
  - Common utility classes.
  - Common exceptions.

- **Web Manager module**: A JEE Web application that includes:
  - Includes a simple Managing web GUI: web pages, navigation logic.
  - Uses the Core module for functions such as:
    - Load Grid Configuration,
    - Interact with grid agents.
- **CLI Manager module**: A java command-line program:
  - Command-line interface: command parsing, text output.
  - Uses the Core module for functions such as:
    - Load Grid Configuration,
    - Interact with grid agents.
- **Any other Manager module**: Any future module that needs to connect to Grid Agents to manage Tomcat instances.

- **Grid Agent module**: Responds to Managers calls and controls local Tomcat instances.
  - Listen to Manager requests.
  - Implements the Grid Agent interfaces.
  - Includes the high-level interaction with Tomcat intances.
  - Defines and uses the Tomcat Management Primitives (as interfaces), but does not implement them.
  - Receives content (deployables, grid configuration changes) and applies them.
- **Grid Agent Primitives for Linux**:
  - Implements the Tomcat Management Primitives for Linux OS.
- **Grid Agent Primitives for Windows**:
  - Implements the Tomcat Management Primitives for Windows OS.
- **Grid Agent Primitives for Mac**:
  - Implements the Tomcat Management Primitives for Mac OS.
- **Grid Agent Primitives for Other**:
  - Implements the Tomcat Management Primitives for Other OS.

The executables themselves are comprised of several modules each that are assembled during the build.

- The Grid Web Manager Executable (a WAR) includes:
  - Core module
  - Web Manager module
- The Grid CLI Manager Executable (a JAR) includes:
  - Core module
  - CLI Manager module
- The Grid Agent Executable (a JAR) includes:
  - Core module
  - Grid Agent module
  - Grid Agent Primitives for Linux
  - Grid Agent Primitives for Windows
  - Grid Agent Primitives for Mac
  - Grid Agent Primitives for Other_

Considering all the above, the following phases could be considered as a base line for the road map of the Tomcat Grid.

## Phase 1 - Core Grid Operation

The first phase includes the most basic features, in order to provide a functioning and useful first version of the Grid.

In particular, no Tomcat instances or Grid Agents automatic provisioning is considered, no configuration GUI (only pre-configured XML config files), no WAR deployments, no command-line interface, no complex grid operations, no secondary managers, and no collections.

*markt: This raises another architectural question. Wouldn't this be more scalable if agents were configured with the location of the primary manager and registered themselves? The manager could persist that registration so an agent would have to be explicitly removed if it was taken off-line permanently.*

*vlad: Definitively. A basic agent install, and registration could "summon" all the necessary software from the primary manager: i.e. Tomcat executables, grid configuration, etc. I thought about something like this as an advanced feature (phase 12), but we can rearrange the priorities if needed.*

Included features are:

1. The Web Grid Manager presents a Web interface that shows information of the whole Grid and present simple buttons to operate the Tomcat instances.
2. The managing logic must be clearly separated from the Web interface logic, since later on, a Command-Line Grid Manager will be included, and will use the same managing logic.
3. The available commands for each instance are:
   - **status**: retrieves the status of a Tomcat instance through the corresponding Grid Agent
   - **trigger-start**: sends a start request to the Tomcat instance using the corresponding Grid Agent
   - **trigger-stop**: sends a stop request to the Tomcat instance using the corresponding Grid Agent
   - **trigger-kill**: sends a kill request to the Tomcat instance using the corresponding Grid Agent
     - *markt: A small thing. I think I'd prefer start/stop/kill/*

        - *vlad: I added these commands on phase 7 and, as I see them, they behave are a little bit different from the trigger ones, specially when we refer to the CLI manager. The **trigger-start**, issues the signal to the Grid Agent and ends. The **start** keeps on working (and updating the user) until the Tomcat instance is actually up (or fails to start up). On the Web interface the trigger start could show up as a simple icon (omitting its name). On the CLI I see the start command as far more useful than the trigger-start.*

4. A simple configuration file lists all the machines and their instances so the Grid knows where each instance resides. [This configuration file is probably in XML format]

5. Grid Agents are installed on each machine and manage all instances in that machine pertaining to the Grid. Grid Agents receive commands from any manager and act accordingly. To manage the instances the Agents use:
    - Shell calls: start an instance, kill an instance.
    - JMX calls to retrieve instance live information.
    - JMX calls to change instance live values, and to request instance shutdown.
    - OS calls for any OS related need.
6. It's assumed that a port will be accessible from each Grid Manager to each machine where the Grid Agents are serving. The firewall, if present must allow active server-type sockets on that port.
    - *markt: Another architectural question. Which end opens the connection, does it stay open and which protocol is used? For example, agents connect to Manager via WebSocket.*
    - *vlad: I always considered the Grid Agent would would open a server NIO socket, to avoid using ephemereal ports. The Grid Agent is always listening, and the Managers connect when needed. In terms of protocol, it could be a ad-hoc one, specially developed for this tool, or use a well-known standard. I have ad-hoc one that I can use, but I'm open to suggestions.*
7. Multiple Grids (and Grid Agents) can be running on the same set (or subset) of machines. If that's the case, Tomcat instances, and Grid Agents run on different ports for each grid. When multiple grids use the same machines they don't interfere with each other and can be operated simultaneously.
8. The status command shows the following information for each instance:
    - Machine
    - Service (a unique grid-wide name for each instance)
    - State
9. The state of an instance can be:

    - **Active**: the instance OS process exists, the instance is serving requests, and it looks healthy [enough].

    - **Zoetic** [for lack of a better word]: the instance OS process exists, but the instance is unresponsive and it doesn't respond to requests for state. It's probably not serving any HTTP requests, does not look healthy, it may be starting, it may be shutting down, it may be overwhelmed. Who knows.

    - **Stopped**: the instance OS process does not exist, and therefore the instance is not operating at all.
    - **Not Available**: This is a pseudo state that the manager applications (web and cli) show when a Grid Agent does not respond to requests for status in a timely manner.
    - If possible it would be great to discern different sub cases of the Zoetic state, so to help the user to determine what's going on and tackle the case accordingly:
        - **Starting**: The Tomcat instance process exists, and the instance is starting. It's not yet serving HTTP requests.
        - **Stopping**: The Tomcat instance process exists, and the instance is stopping. It's no longer serving HTTP requests.
        - **Unresponsive**: The Tomcat instance process exists, but the instance health isn't good, it's not responding to HTTP requests, or it's overwhelmed. It's not even responding to requests for status. This state is different from "Not Available" since in this case the Grid Agent IS active and responding, but the Tomcat instance itself is unresponsive.
        - On second thoughts, these extra states can actually be discerned today with the current version of Tomcat, since the Grid Agents know all the trigger commands each local instance has received and can deduce (or make up) the sub case. If the Grid Agent is restarted, some kind of persistence of its state might be needed to "remember" what was going on before the Grid Agent was shut down, so to make an educated guess.
10. Grid Agents communicate over unsecured TCP sockets, and assume communication security is enforced by the network architecture (segregated segments/VLans).
11. The "trigger"-type commands just deliver the corresponding signal to the instance's Grid Agent and returns right away, without waiting for the full operation to complete. It's kind of "fire and forget". The web user can keep on refreshing the the web interface to find out about the progress of the status of the Tomcat instances.

12. Simple user name/password authentication is implemented to secure the Web interface. [Maybe we'll need to provide more options]

## Phase 2 - Manageable Grid Agents

This phase revamps the Grid Agents so they become manageable.

Included features are:

1. As well as the instances, the agents can become unresponsive, or even crash. To address cases like these commands are implemented to manage the Grid Agents as well. The Grid Agents become now manageable services.
2. All grid agents are now also registered in the configuration file under a unique service names. Grid agents names share the same namespace than Tomcat instance names: i.e. the Tomcat instances & Grid Agent names are grid-wide unique. This way commands (such as a trigger-start for example) can distinguish which type of service it needs to act on, and will chose a different logic (program, or script) to execute.
3. All previously defined commands are now available for the Grid Agents:
    - status
    - trigger-start
    - trigger-stop
    - trigger-kill
4. The status command now adds an extra column "Type" that indicates the type of service: Tomcat instance, or Grid Agent.

5. The mechanism to manage the grid agents is necessarily OS dependent. For example, in Linux it can be implemented using Bash commands though SSH. Suitable mechanisms must be studied for each OS.

## Phase 3. Secondary Managers

The functionality for configuration replication is added.

Included features are:

1. Secondary managers are registered on the grid's configuration file.
2. Every time a configuration change is produced or detected on the configuration of the primary manager, the changes are distributed to all secondary managers.
3. If the primary manager is down, secondary managers can be started, and can start producing changes (based on the local configuration copy). They can also start distributing the new configuration changes.

## Phase 4 - Extended Service & Machine Information

This phase extends the status information of the whole grid beyond the basic data.

Included features are:

1. The status command now adds more information for each service (Tomcat instances and Grid Agents):
    - CPU usage (if possible)
    - CPU load (if possible)
    - Heap usage (if possible)
    - Threads (if possible)
    - Started on (if possible)
    - Any other information deemed useful for managing purposes.
2. [Optional] Machine information (same page, or maybe an extra tab) shows per machine:
    - CPU usage
    - CPU load (1 min, 5, min, 15 min)
    - Memory usage
    - File system space usage for the mount where the "webapps" dir is (can this be different per instance?).

## Phase 5 - Command-Line Interface (CLI)

This phase provides a CLI manager interface for environments that cannot use the web interface.

Included features are:

1. In addition to the Web Grid Manager interface, the Command-Line Grid Manager interface is suitable when the web interface cannot be used. Typical cases are, when no web port is available on the servers (probably fire-walled), when the security policies do not allow remote server operations, etc. This maybe the case on some secured/fire-walled production environments where only text sessions are acceptable.
2. The Command-Line Grid Manager is also suitable for automation (e. g. the weekly full/partial site restart) when unattended operations are scheduled, using cron or equivalent utilities.
3. The Command-Line Grid Manager always leaves a log file per command execution on a directory created for this purpose. Each log file's name includes the time stamp, the command name, and (if possible) the arguments.
4. The implemented commands are:
    - status
    - trigger-start
    - trigger-stop
    - trigger-kill
5. The trigger commands are only executed when necessary. If an instance is already running a trigger-start command will be ignored. Conversely trigger-stop and trigger-kill commands are ignored when the instance is stopped.
6. Return codes must be strategically defined to allow automation. Well defined return codes can provide useful information to the caller program /process (especially for automation), so it can clearly identify the problem and act accordingly.

## Phase 6 - Hooks

Extending the core operation of the grid services with custom logic.

Included features are:

1. Hooks are integration points to include extra activities we want to be performed when some events occur on each Tomcat Instance or Grid Agent. A hook program is linked to a hook and may be implemented as a shell script or any other kind of executable program. Hooks can be defined for the following events:
    - **pre-trigger-start**
    - **post-trigger-start**
    - **pre-trigger-stop**
    - **post-trigger-stop**
    - **pre-trigger-kill**
    - **post-trigger-kill**

2. The hooks are only executed when the corresponding signal is not ignored. For example, if a trigger-start is issued and the instance is stopped, the corresponding pre-trigger-start and post-trigger-start hooks are executed. If the instance was running, then the command would be ignored and its hooks would also be skipped.
3. Hooks can be useful for many purposes. For example, typical uses are:
    - Prepare an instance configuration.
    - Record instance events.
    - Send emails or other notifications upon restarts.
    - Clear caches & temp dirs before starting an instance.
    - Delay the start of an instance to allow the OS to reclaim resources.
    - Generate thread dumps on specific events.
4. Hooks programs run on the machine where the affected instance runs. Therefore, the hooks programs need to be copied and prepared (manually or automagically) to be executed on all machines of the grid.
5. When hooks programs are registered (maybe uploaded) on the Grid they are automatically distributed behind the scenes to all instances /machines before they are ready to be used.

## Phase 7 - Enhanced Grid Operation

Beyond the basic trigger operations, there's usually need for more complex ones that provide very common needs.

Included features are:

1. Non-trigger commands are added to both the Command-Line and Web Grid Managers:
    - **start**: triggers a start and waits until the operation succeeds or fail
    - **stop**: triggers a stop and waits until the operation succeeds or fail
    - **kill**: triggers a kill and waits until the operation succeeds or fail
    - **restart**: triggers a stop, waits until it stops, triggers a start, wait until it starts
    - **killstart**: triggers a kill, waits until it stops, triggers a start, wait until it starts
2. The new commands operate on both types of services (instances and agents).
3. New hooks are added for the new commands:
    - **pre-start**
    - **post-start**
    - **pre-stop**
    - **post-stop**
    - **pre-kill**
    - **post-kill**
    - **pre-restart**
    - **post-restart**
    - **pre-killstart**
    - **post-killstart**
4. All these new commands use the "trigger" primitives behind the scenes.
5. The hooks for the non-trigger events are never ignored, so the hook programs are executed even if the related trigger commands are ignored.
6. Automatic trigger-kill operations are now automatically issued for stop and restart operations if configured, when a trigger-stop fails to succeed in the pre-configured time limit. The time limit is now optionally specified in the configuration file on a per-service basis.
7. A restart delay (now optionally specified on a per-service bases on the configuration file) is used when restarting services: it's applied to the restart and killstart commands.
8. The non-trigger commands show an update of the service state periodically (defaults to every 10s, and can be specified on the configuration file), and they keep working until the full operation completes.

## Phase 8 - Simple Deployment

This phase implements War applications deployment and undeployment to the grid as a whole, or to specific Tomcat instances.

Included features are:

1. New commands:
    - **deploy**: deploys a web application (a WAR) to a specific or all grid instances
    - **undeploy**: undeploys a web application from a specific or all grid instances
2. Through these operations Tomcat instances will be able to run multiple web applications.
3. The status command is revamped so it now lists all war applications deployed on each instance.

## Phase 9 - Application Version Management

The application versioning is like a deployment functionality on steroids.

The Application Version Management may interfere with the simple Deployment as described before, since it manages web applications in a different manner. It needs to be studied if both modes are compatible and can work at the same time, or if both are mutually exclusive. If the latter, the user will need to choose which mode to use when setting up the grid.

When the a new version of the application (a Release) needs to be deployed the deployment happens in an orderly manner. The grid also keeps track of which version is live, which ones are not live but still on the grid, and also provide rollback capabilities.

Included features are:

1. A Release includes one or more deployables. Deployables are WARs, JARs, etc. that will be part of an application we want to deploy on the grid.

2. Maybe all deployables are deployed to all instances, maybe each deployable goes to a subset of instances. This will need to be specified on the configuration file.
3. Releases are first registered into the Grid (maybe even uploaded) under a client-provided unique Version ID. If no Version ID is provided, the system generates it. Behind the scenes, each release deployable is transferred to the corresponding machines automatically during registration. Since this operation may take a while the release will show the states of loading, ready, or removing.
4. Releases must be deployed to the grid using the Version ID. Since all deployables are already distributed to all machines, a deployment now corresponds to local copy (or sym link) of each deployable to the Tomcat's webapps dir. The deployment automatically registers which Version ID is now deployed on every instance. The deployment operation can be sent to the whole grid or to a single instance.
5. No two versions are deployed at the same time on a Tomcat instance. When a version is deployed to an instance, the existing one is first unlinked from the instance. It's not actually removed from the grid or the file system, so a rollback operation can be performed quickly if needed.
6. New commands are implemented:
    - **register**: loads a new application version into the grid under a unique Version ID
    - **deregister**: removes a non-live application version from the grid
    - **versions**: list all loaded and live Version IDs and where they are deployed
    - **deploy**: deploys all version's deployables to the corresponding instances, unlinking the old ones
    - **rollback**: undeploys the current version from all Tomcat instances, and restores the previous version
    - **deploystop**: deploys all version's deployables to the corresponding instances, but keep the Tomcat instances down
    - **undeploy**: undeploys a version (or a single deployable) from the grid or a subset of the grid
7. As shown above the deploy and undeploy commands are revamped.
8. The deploy, rollback, deploystop, and undeploy command may use, behind the scenes, many low-level "link" and "unlink" tasks. These tasks add /remove a deployable to/from an instance correspondingly.
9. Each deploy inspects the current state of the grid and saves a Rollback Plan. The rollback commands executes the Rollback Plan. Only one Rollback Plan is saved at any given time.
10. The Rollback (if saved) is shown on the Web and CLI interfaces.
11. The status operation now also shows for each instance:
    - Deployables (the list of war applications deployed in the instance)
    - Version IDs
    - Deployed at (date & time)
12. New hooks are added for the new commands:
    - **pre-register-release**
    - **post-register-release**
    - **pre-register-deployable**
    - **post-register-deployable**
    - **pre-deregister-release**
    - **post-deregister-release**
    - **pre-deregister-deployable**
    - **post-deregister-deployable**
    - **pre-deploy**
    - **post-deploy**
    - **pre-rollback**
    - **post-rollback**
    - **pre-deploystop**
    - **post-deploystop**
    - **pre-undeploy**
    - **post-undeploy**
    - **unlink** (unlink: low-level operation that removes a deployable from an instance)
    - **post-unlink**
    - **pre-link** (link: low-level operation that adds a deployable to an instance)
    - **post-link**

## Phase 10 - Collections

Collections are groups and sub-groups of services (instances and/or agents) that are managed together. Essentially, instead of issuing a command on a single service, you can now issue it onto a collection. The commands will now affect all services in the collection and will probably take longer to complete, since multiple operations are now needed to complete the whole command.

Collections may group identical Tomcat instances or maybe Tomcat instances that do not serve the same purpose. For example, one subgroup of Tomcat instances may run the customer facing site (like an HTTP cluster), other group may run the back end site (maybe processing JMS queues), other group may be dedicated to serving or connecting to integration points.

Included features are:

1. The following commands can now be issued on collections in addition to plain services:
    - status
    - trigger-start
    - trigger-stop
    - trigger-kill
    - start
    - stop
    - kill
    - restart
    - killstart
    - deploy
    - rollback
    - deploystop
    - undeploy
2. Hooks are modified to provide information of the collection they are affecting.

3. When a hook runs on a collection, it runs on the machine where the manager application (web or cli) runs, not remotely on the machine of any other instance. This is because in this case the execution is not tied to a specific instance, but to a collection.
4. Services defined in a collection can be managed in sequential or parallel modes. For example, a restart command on a sequential collection will restart the second service only, when the first one has fully completed. Once the second completes, it will restart the third one, and so on. A parallel collection would issue a restart on all services simultaneously.
5. Collections are defined in the configuration file and are of a recursive nature: a collection can include plain services, other collections, or both. For sequential mode, each "sub-collection" is treated as a single element so it's considered fully complete when all its included services and collections complete.

6. [To be analyzed and defined if it's useful or not] Collections editing through the Web interface. This can be useful to graphically update collections when machines/instances are added/removed.

## Phase 11 - Instance Configuration

The Web interface adds functionality to specify Tomcat instances configuration from the centralized location.

The CLI interface does not offer this functionality. [to be discussed]

Included features are:

1. Using the Web interface the user can change instance's configuration remotely. This operation allows the instances to be changed remotely, for example to:
    - Add libraries (typically JDBC drivers, MQ drivers, JSF, etc.)
    - Set JVM parameters (memory settings, GC behavior, JVM tweaking, etc.)
    - Prepare (add/change/remove) JDBC data sources
    - Set context parameters
    - Set JNDI entries
    - View required WAR resources, and set resources values accordingly
    - Changing listeners
    - Other instance configurations

## Phase 12 - Instance Provisioning

This functionality removes the necessity of a manual setup of all machines and Tomcat instances of the Grid. After installing the first Tomcat instance and deploying the Web Manager, provisioning operations to local or remote machines can be performed through the web interface.

Because of its nature, the implementation of the provisioning operations is heavily OS dependent.

In addition it may not be possible to install, configure, and run the Grid Agents remotely because of fire-walled machines. If that's the case, the Grid Agents will need to be manually setup. Once the Grid Agents are running, the rest of the provisioning can be performed through the Web Manager, using the Grid Agents.

Included features are:

1. The provisioning operation will automate the following tasks:
    - Login into a machine
    - Installing the Grid Agents
    - Configuring & running the Grid Agent
    - Installing the Tomcat instances
    - Configuring the Tomcat instances
    - Configuring the environment (shell variables, other)
2. Using the Web and Command-Line interfaces the user can provision the Grid. Typical operations can be:
    - Adding a new machine to the Grid.
    - Removing a machine from the Grid.
    - Adding a new instance to a machine.
    - Removing an instance from a machine.
3. Once new machine is registered, the machine's Agent is installed and executed.
4. To deregister a machine all instances must have been removed first.
5. If a machine is deregistered, the Agent is stopped and optionally uninstalled. Maybe we'll leave it there, so it will be easier in the future to re-provision the machine.
6. Once a new instance is created the following operations are performed:
    - Registering the machines on the grid configuration file
    - Standard instance's directory tree is copied
    - All the instance extra configuration (libraries, JDBC data sources, etc.) are performed
    - No deployments are installed yet.
7. To remove an instance, all deployables must have been undeployed first.
8. Once an instance is removed:
    - The instance is removed from the configuration file and any collection that included it
    - All deployments are removed from it
    - The whole directory tree for it is removed on the remote machine
9. The provisioning operations require remote access to the new machine, and therefore some kind of connections needs to be setup. For example, an SSH connection could be used if the user provides the user name/password credentials or if if an ssh key exchange had been previously setup between the machines.

## Phase 13 - Additional Commands

[To be described]