

Centralized coordination

Table of contents

1 Homegeneous versus heterogenous activities.....	2
2 Command execution across like modules.....	2
3 Command execution across heterogeneous objects.....	3

DRAFT: May need corrections!

Deployment activities almost always involve some coordinated set of steps. Activities include package distribution, software re-configuration, and process restarts. These steps may occur locally in a sequence on one host or they may be performed across a set of hosts.

CTL provides an infrastructure and supports a set of patterns to help coordinate deployment and administration steps be they local or distributed.

1. Homegeneous versus heterogenous activities

Some deployment and administration activities impact one kind of thing. For example, you might restart just the httpd servers on a pool of web servers. In contrast the activity may impact various kinds of things related by configuration. Imagine a typical three tier stack of software - Apache Httpd, Tomcat app server, and MySQL RDB - each tier spread out on their own machines.

CTL supports several design patterns to help coordinate either scenario and describes each model as such:

- homogeneous: A homogeneous activity is one where the same action is performed across the same kind of thing (e.g., just HTTP on web hosts).
- heterogeneous: A heterogeneous activity is one where the same action is performed across different kinds of things (e.g., HTTPD, Tomcat and MySQL across all their hosts).

Depending on the scenario, coordination can take on extra dimensions.

2. Command execution across like modules

The simplest coordination example is one you are probably already familiar with if you are an experienced CTL user. In this scenario, the same action is executed across a set of hosts.

In this example, a fictitious "deploy" command residing in a module named "myapp" is invoked via the `controller` task with `nodedispatch` turned on.

```
<command name="deploy" command-type="AntCommand">
  <implementation>
    <controller>
      <execute strategy="nodedispatch">
        <context/>
        <command name="deploy" module="myapp"
depot="${context.depot}"/>
        <nodeset>
          <include tags="${opts.tags}"/>
        </nodeset>
      </execute>
    </controller>
  </implementation>
</command>
```

```
        </execute>
    </controller>
</implementation>
<opts>
    <opt parameter="tags" property="opts.tags" required="true"
type="string"/>
</opts>
</command>
```

All hosts that match the user specified `-tags` argument will have the deploy command executed. This example, also assumes the "myapp" module has been [registered as a deployment](#) on the desired hosts.

3. Command execution across heterogeneous objects

In this scenario, a set of objects distributed across various machines must have the same command performed.

CTL's [apply-macro](#) task is useful to loop over a query result set and execute actions for each result.

The source code sample below shows three object deployment registrations. The `apply-macro` task is used to query for all the objects matching the arguments of `-name` and `-type` and for each one calls the specified `-command`:

```
<command name="dispatch-command" description="dispatch commands."
    command-type="AntCommand" is-static="true">
    <implementation>

        <!--
        ** example deployments.properties data
        -->

        <property name="object-deployment.default.apache.myapp"
value="web1,web2,web3"/>
        <property name="object-deployment.default.tomcat.myapp"
value="app1,app2"/>
        <property name="object-deployment.default.mysql.myapp"
value="rdb1"/>

        <!--
        ** execute the tasks inside the sequential using the results of
the query
        -->
        <apply-macro>
            <propertiesquery
                id="myid"
                select="dType,dName"
```

```

from="object-deployment\.${context.depot}\.(${opts.type})\.(${opts.name})">
    <orderby>
        <select name="dType" by="name" order="ascending"/>
    </orderby>
</propertiesquery>
<sequential>
    <echo>calling ${opts.command} command on object: @${dName}
[/${dType}] ...</echo>

    <controller>
        <execute strategy="nodedispatch">
            <context depot="${context.depot}" entityClass="@${dType}"
entityName="@${dName}"/>
            <command name="${opts.command}"/>
            <nodeset>
                <include names="${opts.hosts}"/>
            </nodeset>
        </execute>
    </controller>

</sequential>
</apply-macro>

</implementation>
<opts>
    <opt parameter="command" property="opts.command" type="string"
required="true"/>
    <opt parameter="hosts" property="opts.hosts" type="string"
required="true" default=".*"/>
    <opt parameter="name" property="opts.name" type="string"
required="true" default=".*"/>
    <opt parameter="type" property="opts.type" type="string"
required="true" default="^[^\.]*"/>
</opts>
</command>

```

To run the status command across all the objects run:

```
ctl -m mymodule -c dispatch-command -- -command status
```

To run the status command across all the apache objects run:

```
ctl -m mymodule -c dispatch-command -- -command status -type apache
```

To run the status command on just the apache object on web1:

```
ctl -m mymodule -c dispatch-command -- -command status -type apache -hosts
web1
```

The command line arguments make the command very flexible and would allow you to build up another level of automation by calling this command via workflow wrapper commands.