Hawk Documentation

Release 2.0.0

Kristoffer Gronlund

Dec 07, 2017

Contents

This guide is an introduction to using Hawk, the High Availability Web Konsole. Hawk is a web interface for the HA Pacemaker stack in Linux. With Hawk, the management and configuration of HA clusters is greatly simplified.

Right now, only a small subset of the features in Hawk are covered, and the Guide is not a complete introduction to HA clusters in general.

Contents:

CHAPTER 1

Installation

This guide comes with a `Vagrantfile` which configures and installs a basic three-node HA cluster running Pacemaker and Hawk. This is the cluster that will be used in the rest of this guide.

Make sure that you have a fairly recent version of Vagrant[1] installed together with either VirtualBox or libvirt as the VM host. To use libvirt, you may need to install the bindfs[3] plugin for Vagrant. For more details on how to install and use Vagrant, see https://www.vagrantup.com/ .

To begin setting up the example cluster, use `git` to check out a copy of the source repository for this guide:

```
$ git clone --recursive git@github.com:krig/hawk-guide
```

Now let Vagrant configure a virtual machine[2] running Hawk:

```
$ cd hawk-guide
$ vagrant up alice
```

If everything goes as it should, Vagrant will go off and do its thing, downloading a base VM image and installing all the necessary network configuration and software packages that we'll need. Once the installation finishes, a VM running Pacemaker and Hawk should start up.

The `Vagrantfile` can configure three VMs: `alice`, `bob1` and `bob2`. So far we've only configured `alice`, but once you have confirmed that the installation was successful you can also start the other two VMs using `vagrant up`:

```
$ vagrant up bob1
$ vagrant up bob2
```

Make sure Hawk is running by logging into the `alice` VM and running the following commands:

[1] https://www.vagrantup.com/

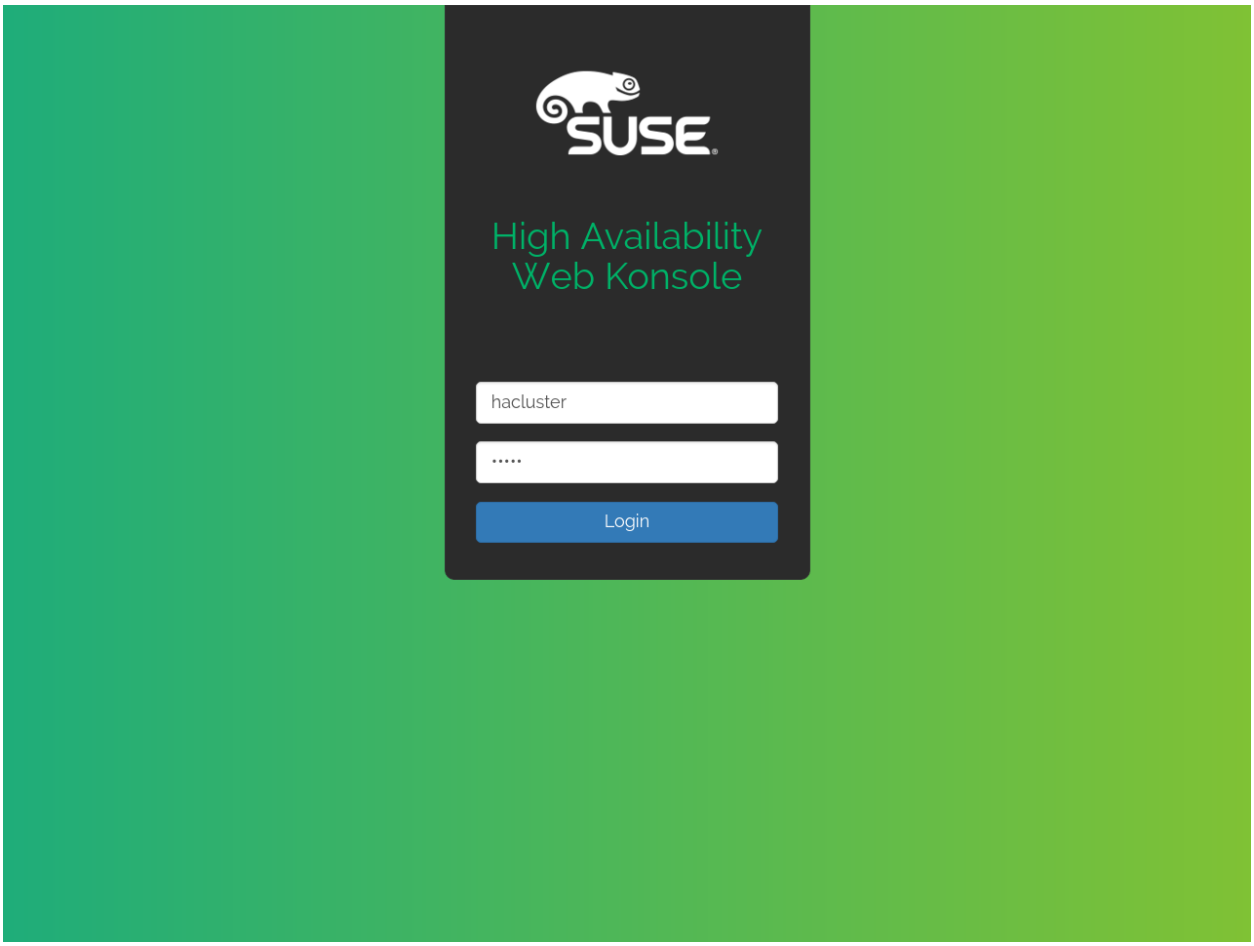
[3] https://github.com/gael-ian/vagrant-bindfs

[2] This command lets Vagrant decide which virtualization provider to use. To select a provider manually, pass `--provider=libvirt|virtualbox|...` parameter to `vagrant up`.

```
$ vagrant ssh alice
(alice) $ sudo chkconfig hawk on
(alice) $ sudo service hawk start
```

1.1 Logging in

To view the Hawk web interface, open this URL in your web browser: https://localhost:7630/



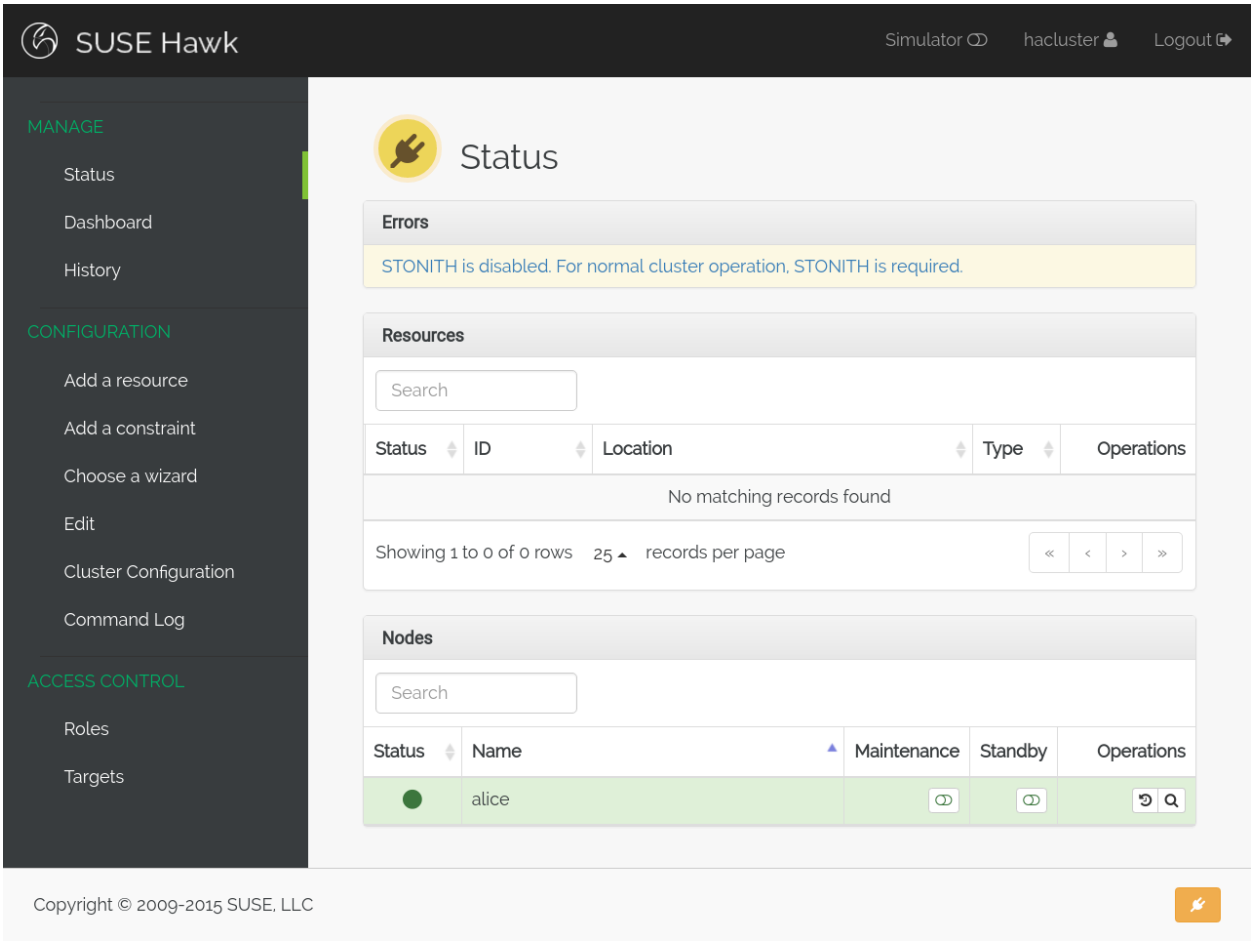
Connecting to port `7630` on `localhost` should work for the above described installation method, since the `Vagrantfile` also forwards port `7630` from the virtual machine. If you installed Hawk on a different machine than the one you are currently using, you will need to connect to port `7630` on that machine instead.

You may see a prompt warning you about an unverified SSL certificate. By default, Hawk generates and presents a self-signed certificate which browsers are understandably sceptical about.

Once you have accepted the certificate, you will be faced with a username and password prompt. Enter `hacluster` as the username and `linux` as the password. This is the default identity as configured by the HA bootstrap script. Naturally, you would want to change this password before exposing this cluster to the wider world.

1.2 A note on fencing

After logging in, the Status view of Hawk should appear, looking something like the image below.



Notice that the status is yellow (meaning warning), and that there is an error reported:

```
STONITH is disabled. For normal cluster operation, STONITH is required.
```

STONITH, or fencing, is an essential element in any production cluster. When a node stops communicating or gives conflicting information to its peers, the other nodes need some way to ensure that the misbehaving node isn't running resources it shouldn't be. Examples of this kind of failure are network outages or a failed stop action.

The mechanism used by Pacemaker to handle these kinds of failure is usually referred to as fencing or STONITH[4].

Any production cluster needs fencing. However, fencing can be complex to configure, especially in an automatic fashion. There are fencing agents available for both libvirt and VirtualBox, and there is also a form of fencing which relies on shared storage called SBD[5].

To learn how to configure an actual fencing device for this cluster and get rid of that warning, see *Configuring fencing*.

[4] Shoot the Other Node in the Head.
[5] https://github.com/ClusterLabs/sbd

CHAPTER 2

Basic Concepts

Before we get any further, we should establish some basic concepts and terminology used in High Availability.

Cluster A cluster in the sense used in High Availability consists of one or more communicating computers, either virtual machines or physical hardware. It's possible to mix and match virtual and physical machines.

Node A node is a single machine participating in a cluster. Nodes invariably fail or experience malfunction. The HA software provides reliable operations by connecting multiple nodes together, each monitoring the state of the others, coordinating the allocation of resources across all healthy nodes.

Resource Anything that can be managed by the cluster is a resource. Pacemaker knows how to manage software using LSB init scripts, systemd service units or OCF resource agents. OCF is a common standard for HA clusters providing a configurable interface to many common applications. The OCF agents are adapted for running in a clustered environment, and provide configurable parameters and monitoring functionality.

Constraint Constraints are rules that Pacemaker uses to determine where and how to start and stop resources. Using constraints, you can limit a resource to run only on a certain subset of nodes or set a preference for where a resource should normally be running. You can also use more complex rule expressions to move resources between nodes according to the time of day or date, for example. This guide won't go into all the details of what can be done with constraints, but later on we will create and test constraints using Hawk.

CIB Cluster Information Base. This is the configuration of the cluster, which is configured in a single location and automatically synchronised across the cluster. The format of the configuration is XML, but usually there is no need to look at the XML directly. The CRM Shell provides a line-based syntax which is easier to work with from the command line, and Hawk provides a graphical interface with which to work with the configuration.

CRM Shell Behind the scenes, Hawk uses the CRM command shell to interact with the cluster. The CRM shell can be used directly from the command line via the command `crm`.

CHAPTER 3

Configuring fencing

STONITH, or fencing[1], is the mechanism by which Pacemaker makes sure that nodes that misbehave don't cause any additional problems. If a node that was running a given resource stops communicating with the other nodes in the cluster, there is no way to know if that node is still running the resource and for whatever reason is just unable to communicate, or if it has crashed completely and the resource needs to be restarted elsewhere.

In order to make certain what is uncertain, the other nodes can use an external fencing mechanism to cut power to the misbehaving node, or in some other way ensure that it is no longer running anything.

There are many different fencing mechanisms, and which agent to use depends strongly on the type of nodes that are part of the cluster.

For most virtual machine hosts there is a fencing agent which communicates with the hypervisor, for example the `fence_vbox` or `external/libvirt` agents. For physical hardware, the most general fencing device is called SBD, and relies on shared storage like a SAN.

3.1 external/libvirt

There are several different fencing agents available that can communicate with a libvirt-based hypervisor through the `virsh` command line tool. In this example, fencing device of choice is the `stonith:external/libvirt` agent. This ships as part of the `cluster-glue` package on openSUSE, and is already installed on the cluster nodes.

To ensure that communication between the cluster nodes and the hypervisor is authenticated, we need the SSH key of each node to be authorized to access the hypervisor.

In the example cluster, Vagrant has already created an SSH key for us. If you do not have an ssh key, you will need to run the `ssh-keygen` command as `root` on each node:

```
$ ssh-keygen -t rsa
```

Once the SSH keys have been created, execute the following command as `root` on each of the cluster nodes:

[1] The two terms come from the merging of two different cluster projects: The Linux HA project traditionally uses the term STONITH, while the Red Hat cluster suite uses fencing to denote the same concept.

```
$ ssh-copy-id 10.13.38.1
```

Replace `10.13.38.1` with the hostname or IP address of the hypervisor. Make sure that the hostname resolves correctly from all of the cluster nodes.

Before configuring the cluster resource, lets test the fencing device manually to make sure it works. To do this, we need values for two parameters: `hypervisor_uri` and `hostlist`.

For `hypervisor_uri`, the value should look like the following:

```
qemu+ssh://<hypervisor>/system
```

Replace `<hypervisor>` with the hostname or IP address of the hypervisor.

Configuring the `hostlist` is slightly more complicated. Most likely, the virtual machines have different names than their hostnames.

To check the actual names of your virtual machines, use `virsh list` as a privileged user on the hypervisor. This is what the output can look like:

```
 Id    Name                           State
----------------------------------------------------
 4     hawk-guide_alice               running
 7     hawk-guide_bob1                running
 8     hawk-guide_bob2                running
```

If the names of the virtual machines aren't exactly the same as the hostnames `alice`, `bob1` and `bob2`, you will need to use the longer syntax for the `hostlist` parameter:

```
hostlist="alice[:<alice-vm-name>],bob1[:<bob1-vm-name>],..."
```

Replace `<alice-vm-name>` with the actual name of the virtual machine known as `alice` in the cluster. If the virtual machines happen to have the same name as the hostname of each machine, the `:<vm-name>` part is not necessary.

With this information, we can reboot one of the Bobs from Alice using the `stonith` command as `root`:

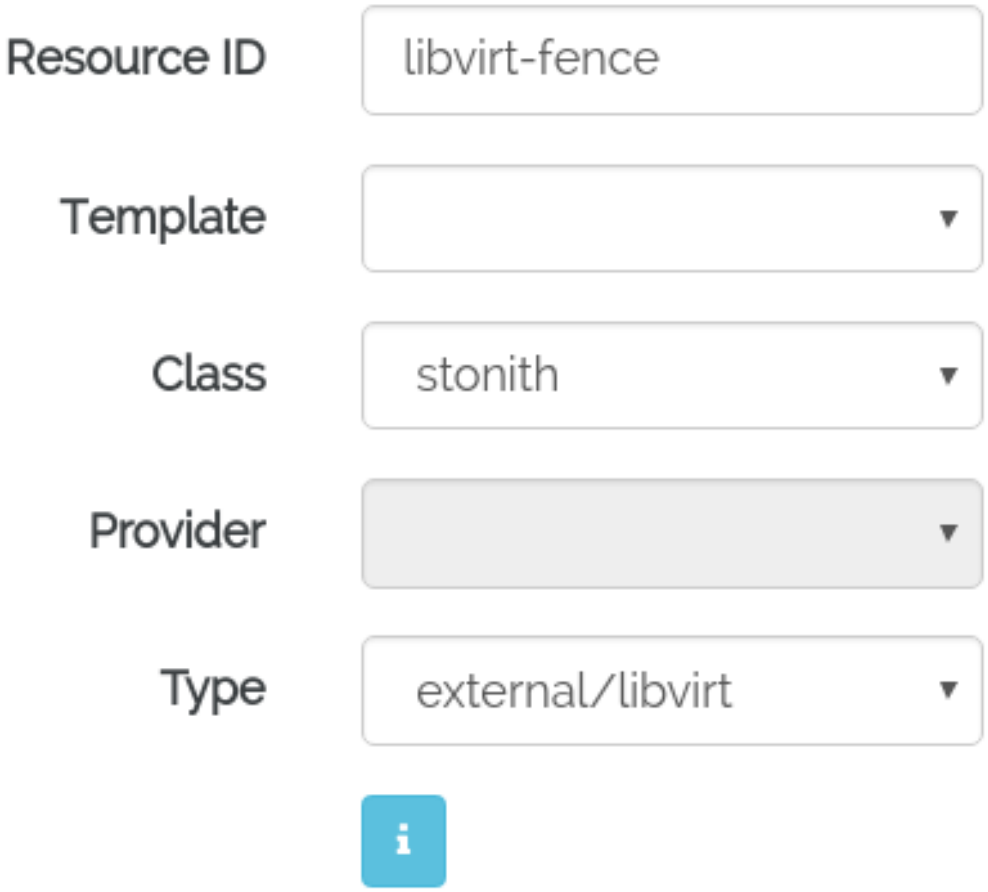
```
$ stonith -t external/libvirt \
    hostlist="alice:hawk-guide_alice,bob1:hawk-guide_bob1,bob2:hawk-guide_bob2" \
    hypervisor_uri="qemu+ssh://10.13.38.1/system" \
    -T reset bob1
```

If everything is configured correctly, this should be the resulting output:

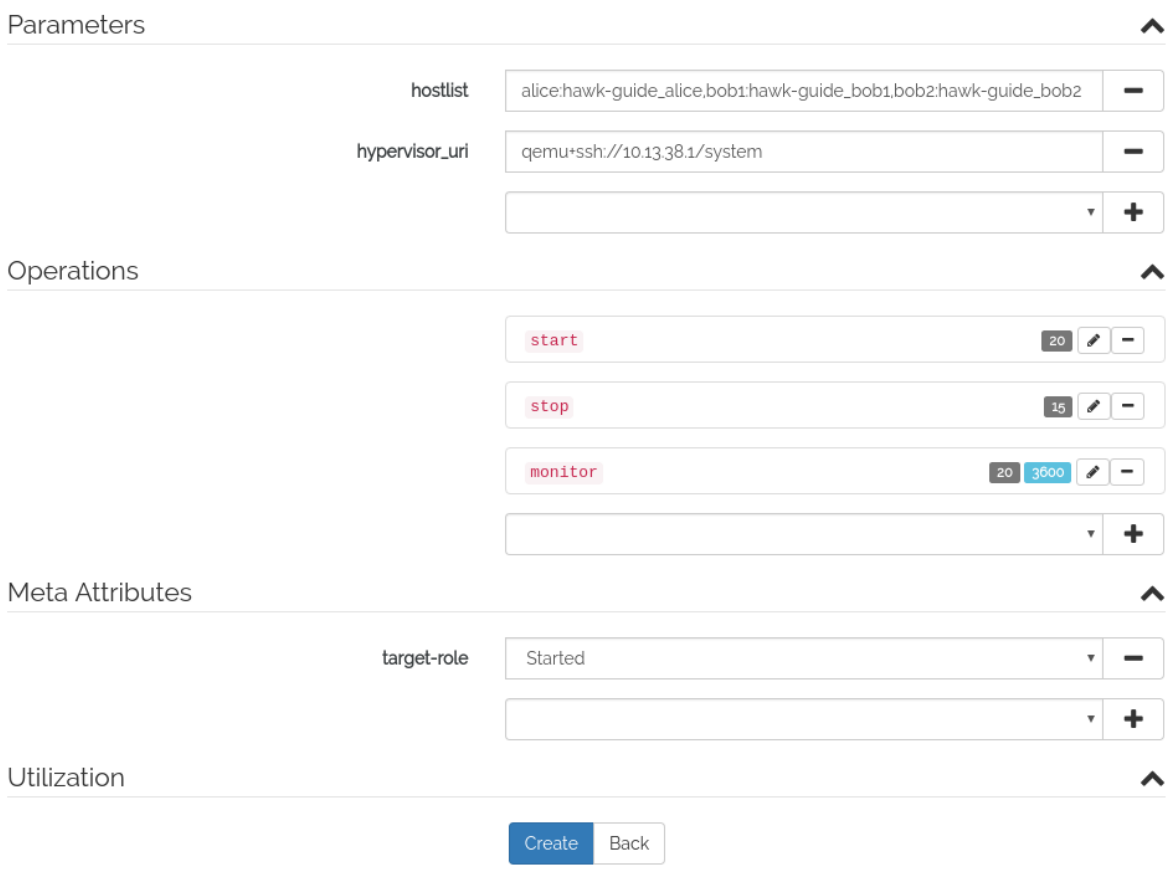
```
external/libvirt[23004]: notice: Domain hawk-guide_bob1 was stopped
external/libvirt[23004]: notice: Domain hawk-guide_bob1 was started
```

Once the fencing configuration is confirmed to be working, we can use Hawk to configure the actual fencing resource in the cluster.

1. Open Hawk by going to https://localhost:7630 and select *Add a Resource* from the sidebar on the left.
2. Click *Primitive* to create a new primitive resource.
3. Name the resource `libvirt-fence` and in the selection box for Class, choose `stonith`. The *Provider* selection box will become disabled. Now choose `external/libvirt` in the *Type* selection box.



4. For the `hostlist` and `hypervisor_url` parameters, enter the same values as were used when testing the agent manually above.
5. Change the `target-role` meta attribute to `Started`.



6. Click the *Create* button to create the fencing agent.
7. Go to the *Cluster Configuration* screen in Hawk, by selecting it from the sidebar. Enable fencing by setting `stonith-enabled` to `Yes`.

A note of caution: When things go wrong while configuring fencing, it can be a bit of a hassle. Since we're configuring a means of which Pacemaker can reboot its own nodes, if we aren't careful it might start doing just that. In a two-node cluster, a misconfigured fencing resource can easily lead to a reboot loop where two cluster nodes repeatedly fence each other. This is less likely with three nodes, but be careful.

3.2 fence_vbox (VirtualBox) [TODO]

The fence agent for clusters using VirtualBox to host the virtual machines is called `fence_vbox`, and ships in the `fence-agents` package.

The `fence_vbox` fencing agent is configured in much the same way as `external/libvirt`.

TODO

3.3 external/ec2 (Amazon EC2) [TODO]

The `external/ec2` fence agent provides fencing that works for cluster nodes running in the Amazon EC2 public cloud.

1. Install the AWS CLI. For instructions on how to do this, see the Amazon start guide: http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html
2. Create the fence resource with the following commands (replacing `<node>` and `<tag>` with appropriate values for your cluster:

```
$ crm configure primitive fencing-<node> stonith:external/ec2 \
    params \
      pcmk_off_timeout="300s" \
      port="<node>" \
      tag="<tag-name>" \
    op start interval="0s" timeout="60s" \
    op monitor interval="3600s" timeout="60s" \
    op stop interval="0s" timeout="60s"
 $ crm configure location loc-fence-<node> \
    fencing-<node> -inf: <node>
```

It is necessary to create a separate fence resource for each node in the cluster. The location constraint ensures that the fence resource responsible for managing node A never runs on node A itself.

TODO: Verify these instructions, use Hawk to configure the resource.

3.4 SBD [TODO]

SBD[2] can be used in any situation where a shared storage device such as a SAN or iSCSI is available. It has proven to be more reliable than many firmware fencing devices, and is the recommended method for fencing physical hardware nodes.

There are two preparatory steps that need to be taken before configuring SBD:

1. Ensure that you have a **watchdog device** enabled. Either this is available depending on your platform, or you would use the software watchdog that the Linux kernel provides. Note that use of the software watchdog makes SBD less reliable than with a true watchdog device.
2. Set up a shared storage device. This needs to be writable by all nodes. It can be very small, SBD only needs about 1MB of space, but it cannot be used for anything other than SBD.

Once a watchdog is enabled and all cluster nodes can access the shared block device, SBD can be enabled and configured as a cluster resource:

1. Configure SBD using the */etc/sysconfig/sbd* configuration file. For details on how to configure SBD, see the SBD man page: https://github.com/ClusterLabs/sbd/blob/master/man/sbd.8.pod
2. Enable the SBD service on each cluster node:

```
$ systemctl enable sbd
$ systemctl start sbd
```

3. Configure the SBD cluster resource:

[2] Shared-storage Based Death. https://github.com/ClusterLabs/sbd

```
$ crm configure \
    primitive fencing stonith:external/sbd \
    pcmk_delay_max=30
```

TODO: Verify these instructions, use Hawk to configure the resource.

CHAPTER 4

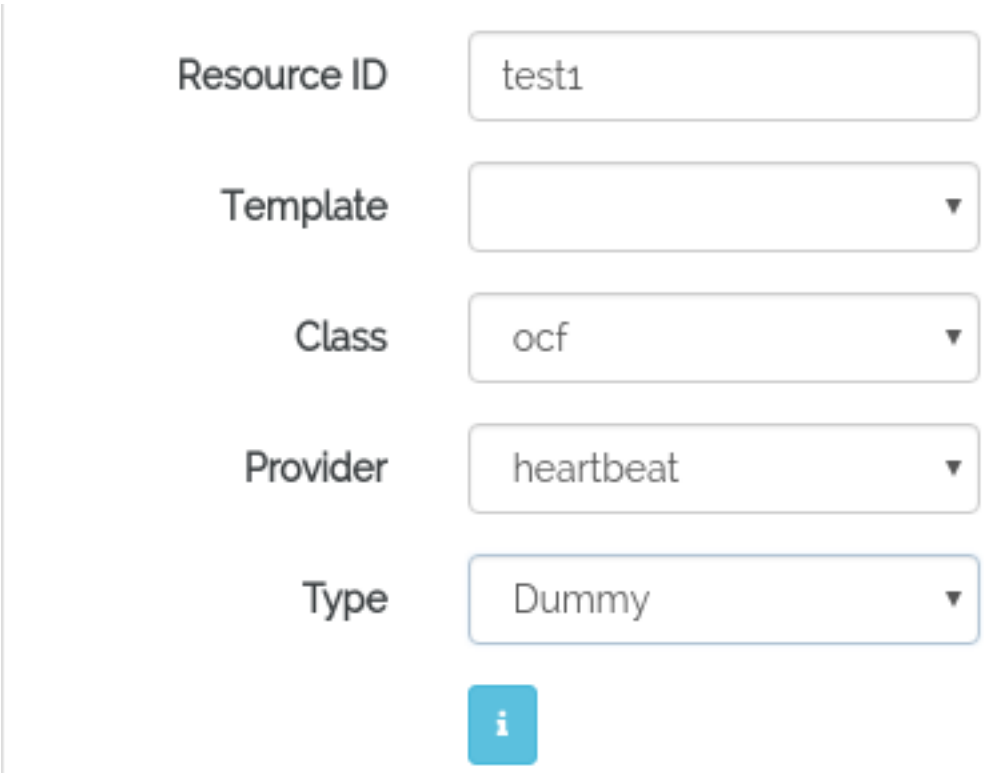
Creating a Resource

To learn how to control resources in the cluster without having to worry about an actual application, we can use the aptly named `Dummy` resource agent. In this section of the guide, we will create a new dummy resource and look at the status and dashboard views of Hawk, to see how we can start and stop the resource, reconfigure parameters and monitor its location and status.

4.1 Add a resource

To add a resource, click *Add Resource* in the sidebar on the left. This brings up a list of different types of resources we can create. All basic resources that map to a resource agent or service are called *Primitive resources*. Click the **Primitive** button on this screen.

1. Name the resource `test1`. No need to complicate things.
2. From the **Class** selection box, pick `ocf`. The **Provider** selection box will default to `heartbeat`. This is what we want in this case. Other resource agents may have different providers.
3. From the **Type** selection box, select `Dummy`. To learn more about the Dummy resource agent and its parameters, you can click the blue *i* button below the selection boxes. This brings up a modal dialog describing the selected agent.



4.1.1 Parameters

The Dummy agent does not require any parameters, but it does have two: `fake` and `state`. In the selection box under *Parameters* it is possible to select either one of these, and by clicking the plus next to the selection box, the parameter can be configured with a value.

On the right-hand side of the screen, documentation for the parameter is shown when highlighted.

For now, there is no need to set any value here. To remove a parameter, click the minus button next to it.

4.1.2 Operations

In order for Pacemaker to monitor the state of the resource, a `monitor` operation can be configured. Resources can have multiple operations, and each operation can be configured with parameters such as timeout and interval. Hawk has configured some reasonable default operations, but in many cases you will need to modify the timeout or interval of your resource.

If no `monitor` operation is configured, Pacemaker won't check to see if the application it maps to is still running. Most resources should have a `monitor` operation.

4.1.3 Meta Attributes

Meta attributes are parameters common to all resources. The most commonly seen attribute is the `target-role` attribute, which tells Pacemaker what state the resource ought to be in. To have Pacemaker start the resource, the `target-role` attribute should be set to `Started`. By default, Hawk sets this attribute to `Stopped`, so that any necessary constraints or other dependencies can be applied before trying to start it.

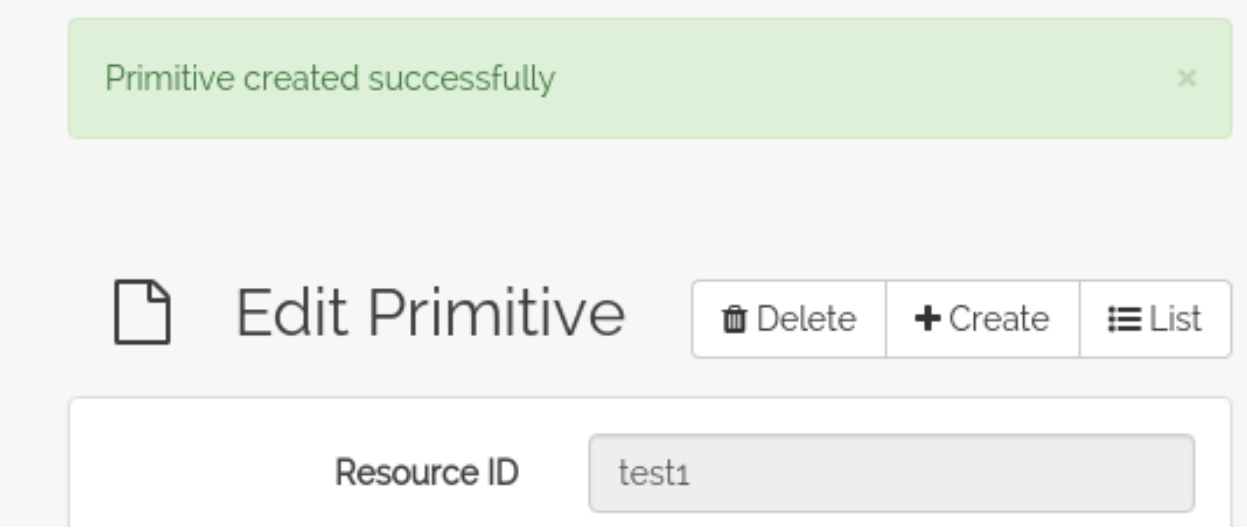
In this case, there are no dependencies, so set the value of this attribute to `Started`.

4.1.4 Utilization

Utilization can be used to balance resources across different nodes. Perhaps one of the nodes has more RAM or disk space than the others, or perhaps there is a limit on how many resources can run on a given node. The utilization values can be used to manage this. By configuring utilization limits on the nodes themselves and configuring utilization values on the resources, resources can be balanced across the cluster according to the properties of nodes.

4.1.5 Finishing Up

To complete the configuration of this dummy resource, click Create. Hawk will post a notification showing if it completed the operation successfully or not.

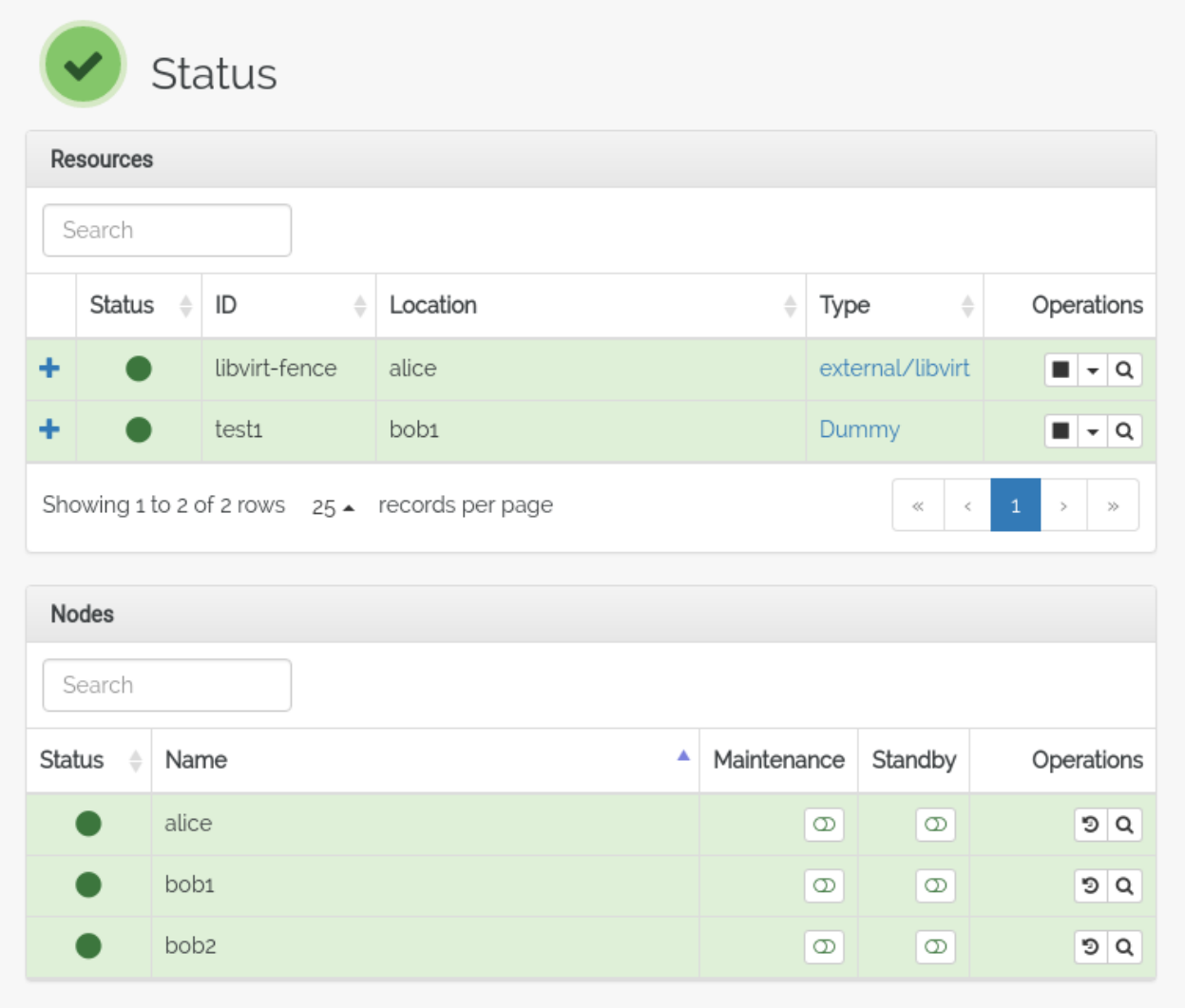


4.1.6 Command Log

To see the command used to create the resource, go to the *Command Log* in the sidebar to the left of the screen. Here you can see a list of `crm` commands executed by Hawk, with the most recent command listed first.

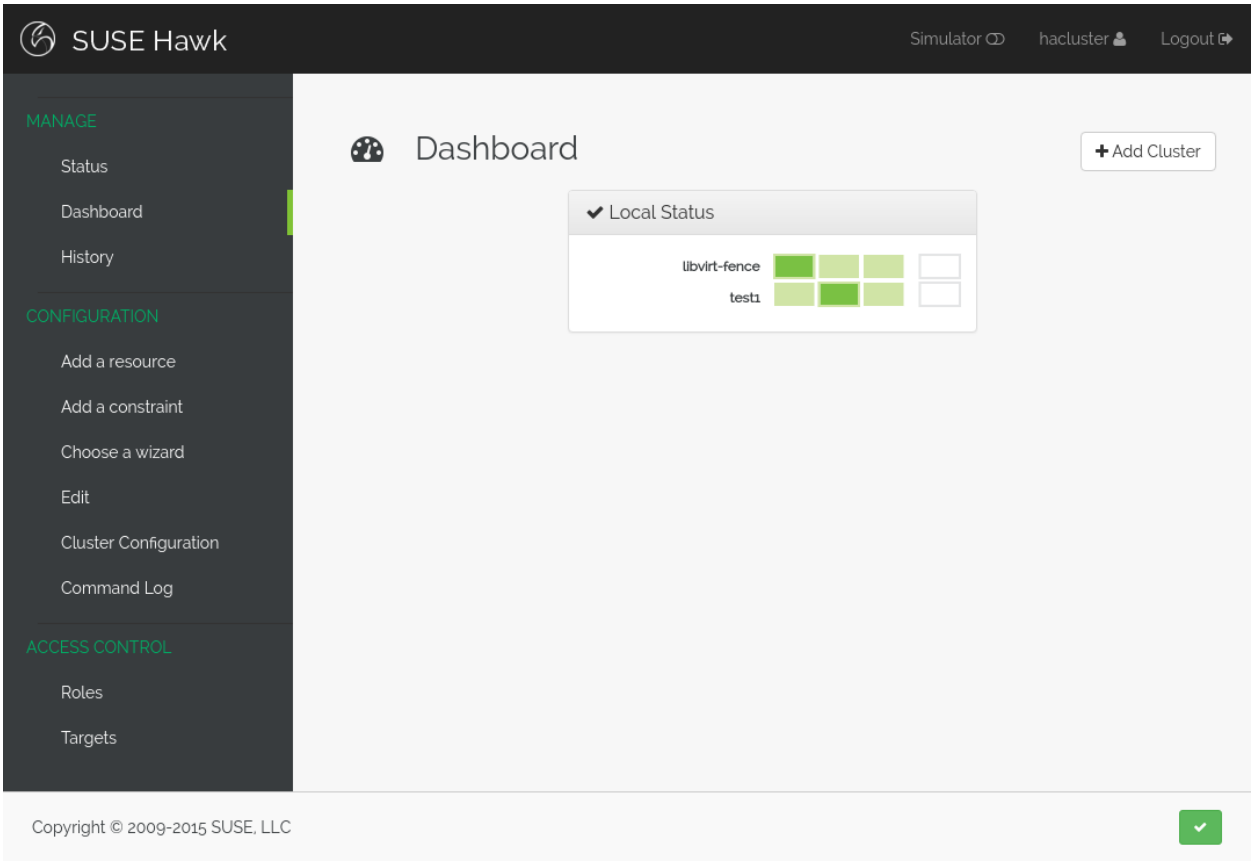
4.2 Status and Dashboard

The created resource `test1` will appear as a green line in the Hawk status view. Stopped resources are colored white, while failed resources are red.



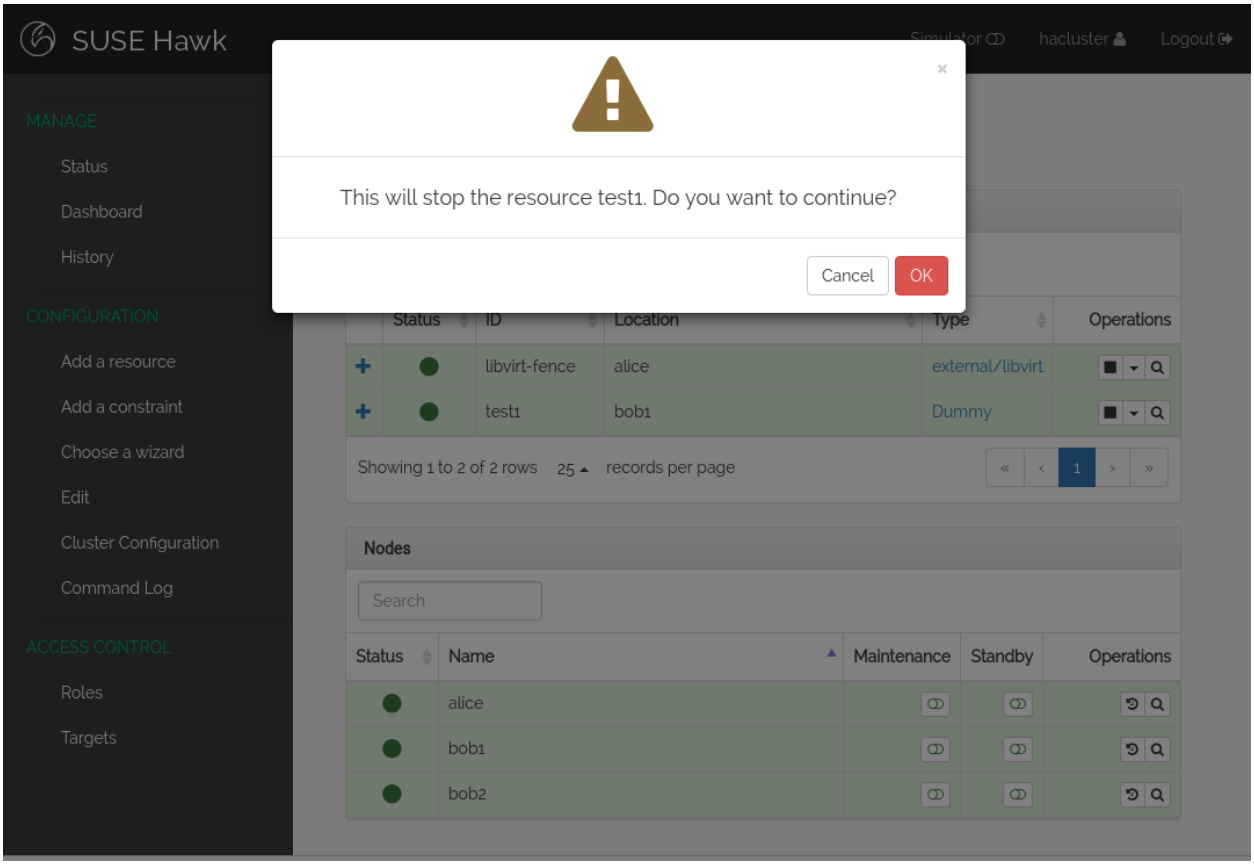
The Dashboard view gives an alternative view of the cluster status. In this view, the cluster is represented by a matrix of lights indicating the state of the resource on each node.

Each row is a resource, and each column is a node. The rightmost column holds resources that are stopped and therefore not running on any node.



4.2.1 Starting and Stopping

Resources can be started and stopped directly from the status view. Use the control icons to the right of the resource listing. When stopping a resource, Hawk will ask for verification before applying the change.



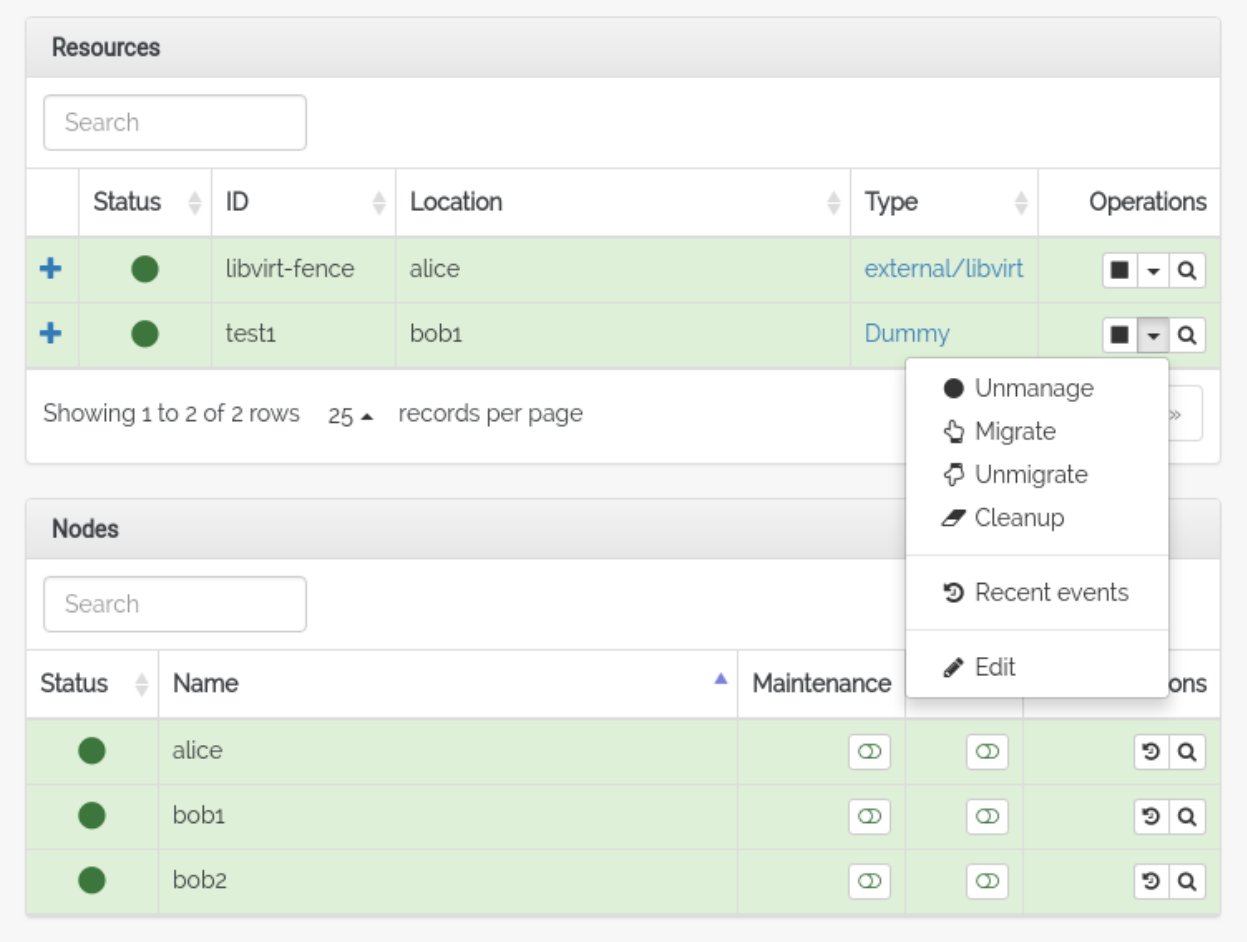
Try stopping and starting the resource. Open the Dashboard in another browser window and see how it updates when the resource is stopped or started.

4.2.2 Migrating Resources

Clicking the down arrow icon next to a resource opens the extended operation menu. From this menu you can choose from a list of more advanced resource operations.

Migrating a resource means moving it away from its current location. This is done with the help of a location constraint created by the user interface. Migration can be given a destination node as an argument, or if no node is provided, the resource is migrated to any other node.

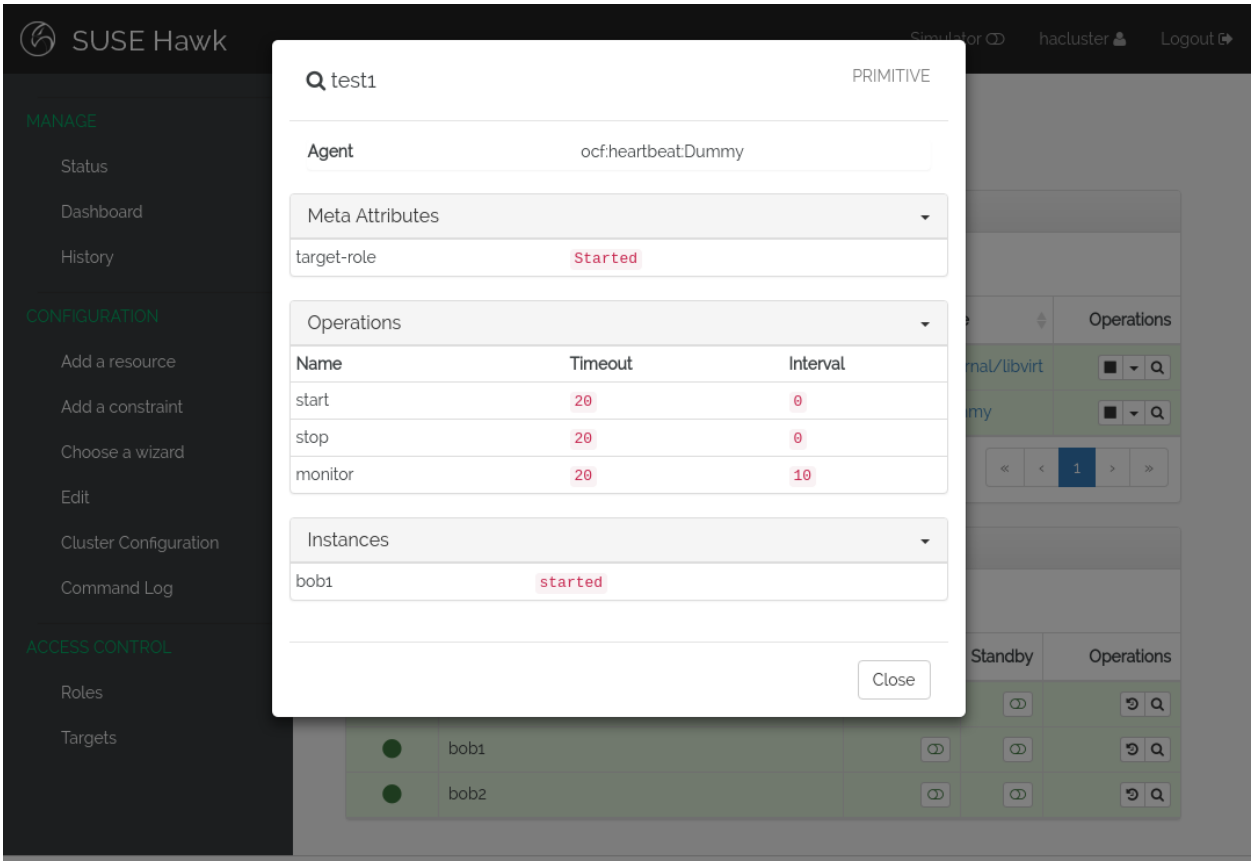
To create such a migration constraint, use the *Migrate* action in the resource menu.



Once the resource has been migrated, the constraint can be removed. This is done using the *Unmigrate* action. Note, however, that once the constraint is removed, Pacemaker may decide to move the resource back to its original location. To prevent this from happening, set the `resource-stickiness` cluster property.

4.2.3 Details

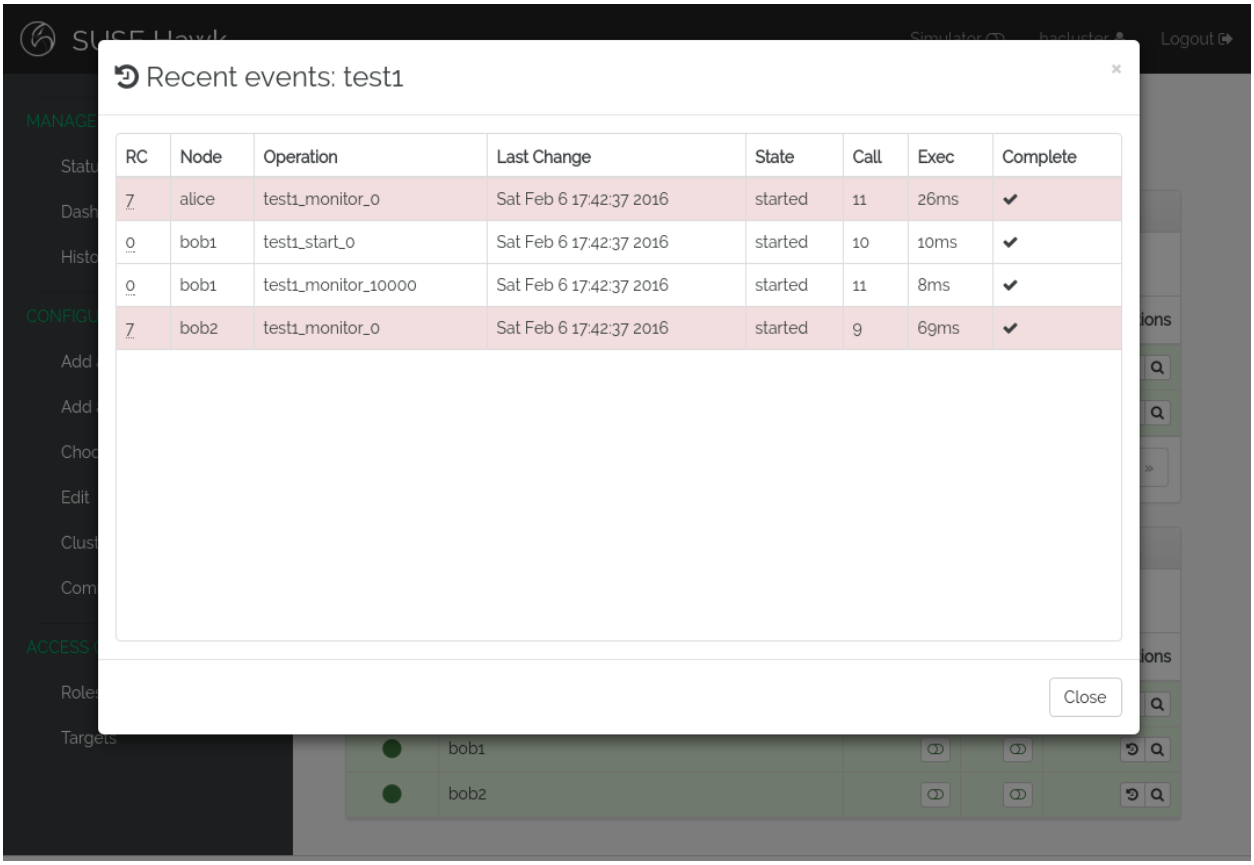
The Details view is accessed via the looking glass button for a resource. This view shows the resource configuration and other details, plus a list of instances.



4.2.4 Recent Events

The Recent Events pane shows a list of actions taken by Pacemaker related to the resource. Each action has a return code, the meaning of which is explained by the tooltip which shows when hovering the mouse over the code. For example `0` means success, while `7` means that the resource was not running.

In the example view, you can see multiple red lines indicating that the resource action failed. These are not actual failures. Pacemaker runs monitor actions for resources on **all** nodes in the cluster, to make sure that the resource is not running where it shouldn't be. These probes show up as failed actions in the Recent Event view, but they are in fact expected to fail.



It is possible to disable these probes for a resource using the *Resource Discovery* attribute on a location constraint. This, however, is generally not a good idea and is only needed for some specific advanced configurations.

4.3 Editing resources

The Edit view for a resource can be found either through the operation menu on the status view, or through the *Edit* screen accessible from the sidebar on the left.

Once in the edit view, you can change any parameters or attributes for the resource, or even delete it.

Note that it is not yet possible to change the resource type of an existing resource in Hawk.

4.3.1 Renaming

To rename a resource, go to the *Edit Configuration* screen, and use the *Rename* operation for the resource.

Only stopped resources may be renamed.

CHAPTER 5

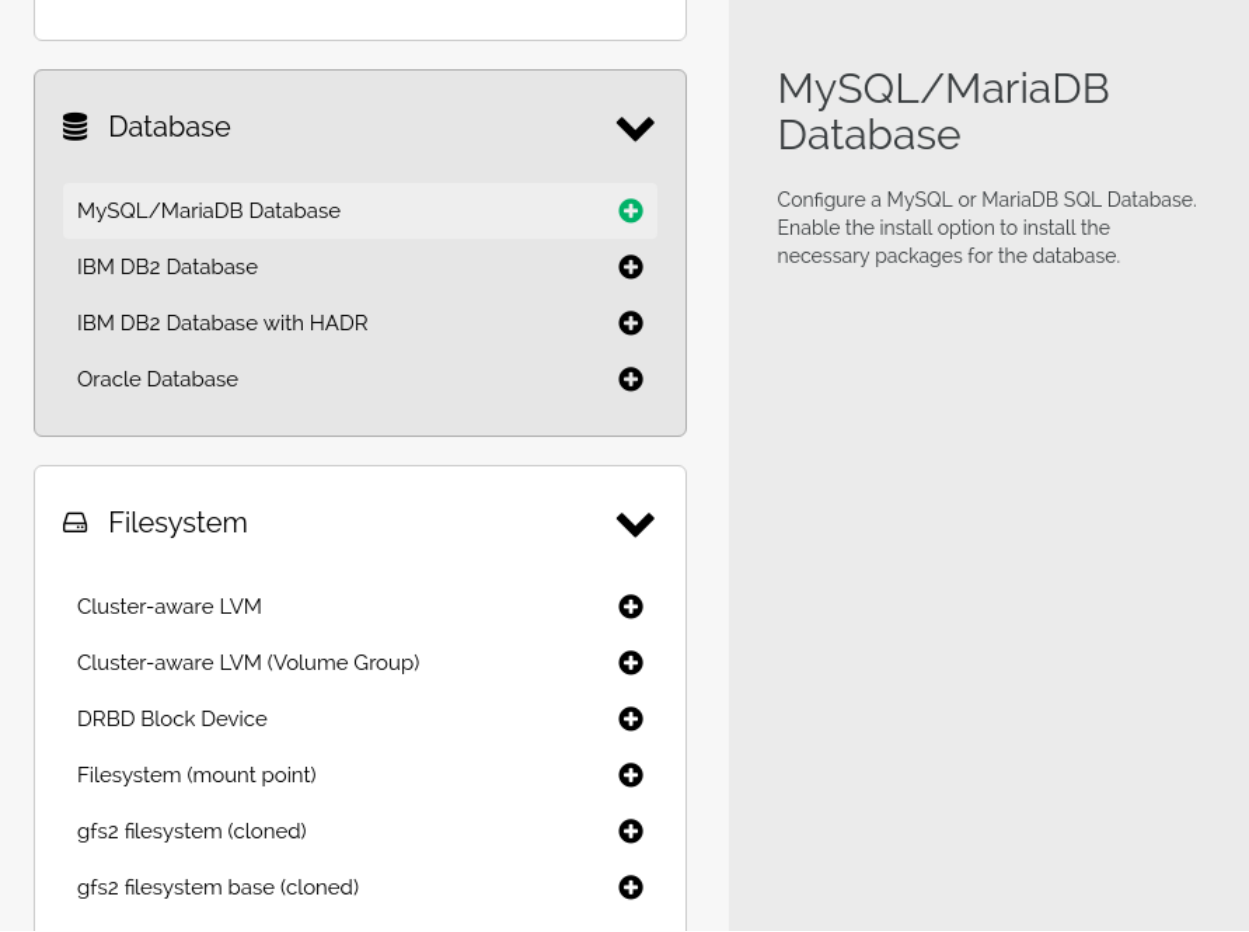
Wizards

A guide to using a wizard to configure cluster resources.

TODO: Expand this into an actual guide using the wizards to configure real resources.

5.1 Select a Wizard

On the wizard selection screen are a list of categories. In each category there is a list of wizards, for example the MariaDB wizard in the Database category.



Each wizard comes with a set of steps, some of which may be optional. In the end, the wizard will verify any parameters and present a list of actions that will be performed on the cluster. This may involve creating new resources or constraints, but also package installation, configuration and verification.

5.2 Setting Parameters

Each step has a list of parameters to set. Some may have default values, while others need to be configured. In some cases, the wizard presupposes the existance of certain resources. For example, the file system wizards generally require that the file system to be managed by the cluster already exists.

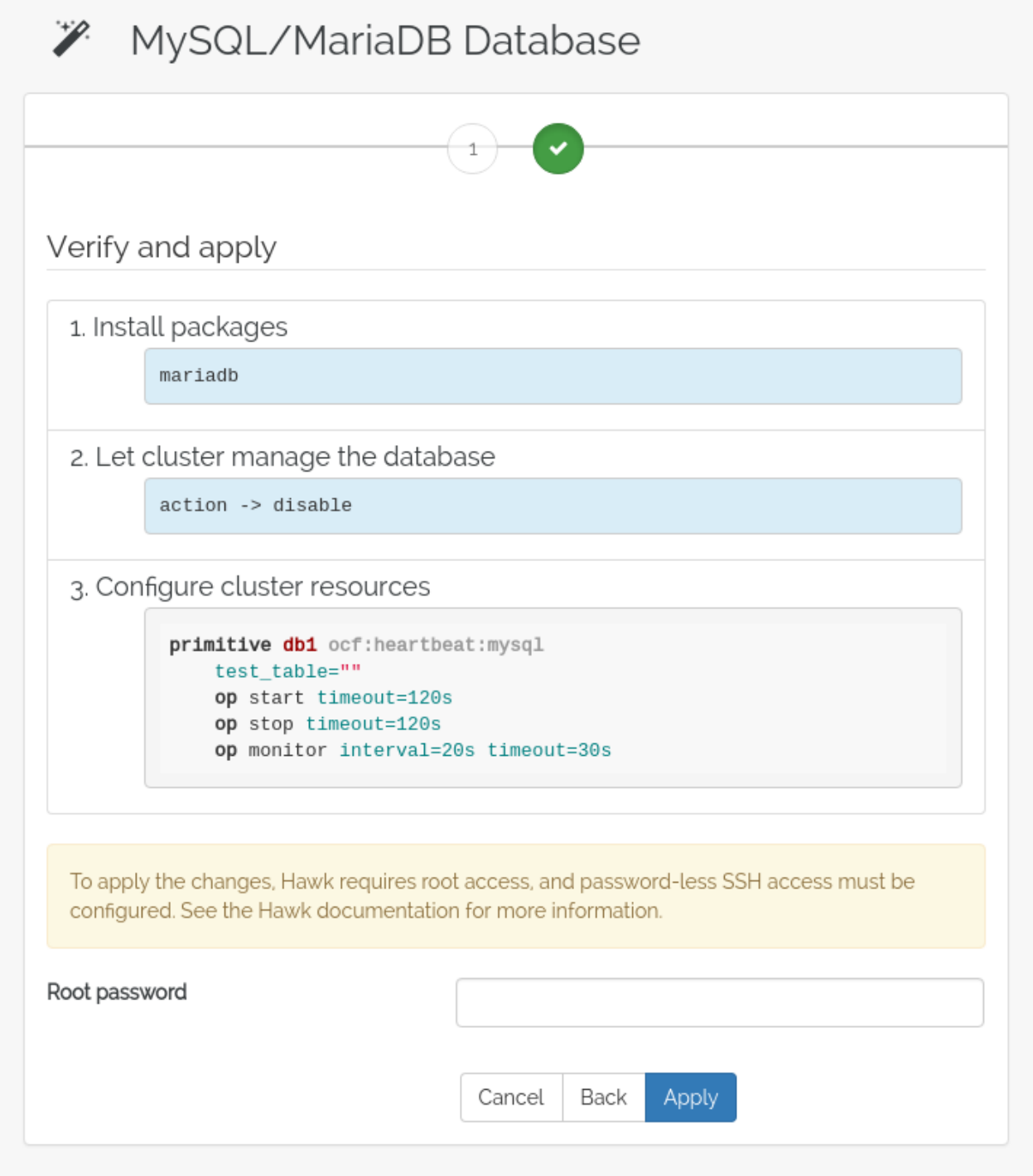
Some wizard steps may have a lot of possible parameters. These parameters will be listed in the *Advanced* section. Most of the time, there should be no need to delve into this section. However, sometimes it can be useful to be able to change a more obscure parameter of a wizard.

5.3 Optional Steps

Some wizards have optional steps, for example configuring a virtual IP that can be used together with a resource. These can be skipped, and whatever actions they would perform will be left out.

5.4 Verify and Apply

Once all the parameters for a wizard have been filled out, the wizard can be applied. If the wizard needs root access on the cluster nodes, for example if it needs to install any packages, the wizard will prompt for the root password here.



Applying the wizard may take a while. Once it completes, a notification will indicate if the wizard was successful.

CHAPTER 6

History Explorer

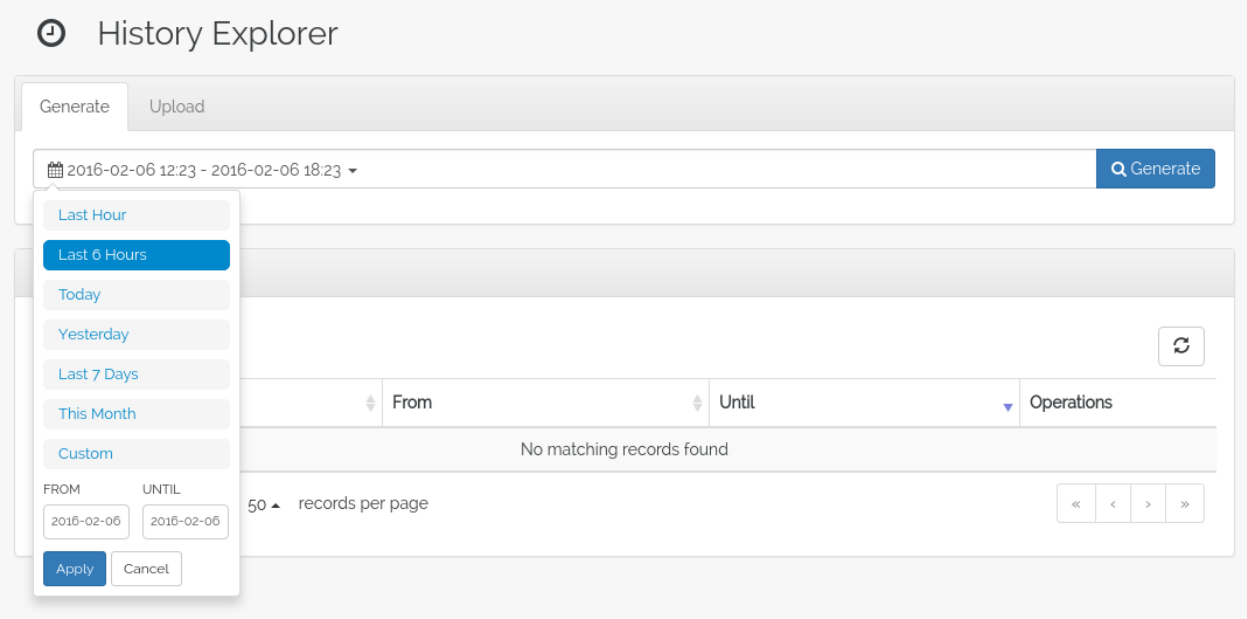
TODO: Describe the history explorer, example usage.

The history explorer is a tool for collecting and downloading cluster reports, which include logs and other information for a certain timeframe. The history explorer is also useful for analysing such cluster reports. You can either upload a previously generated cluster report for analysis, or generate one on the fly.

Once uploaded, you can scroll through all of the cluster events that took place in the time frame covered by the report. For each event, you can see the current cluster configuration, logs from all cluster nodes and a transition graph showing exactly what happened and why.

6.1 Create / Upload

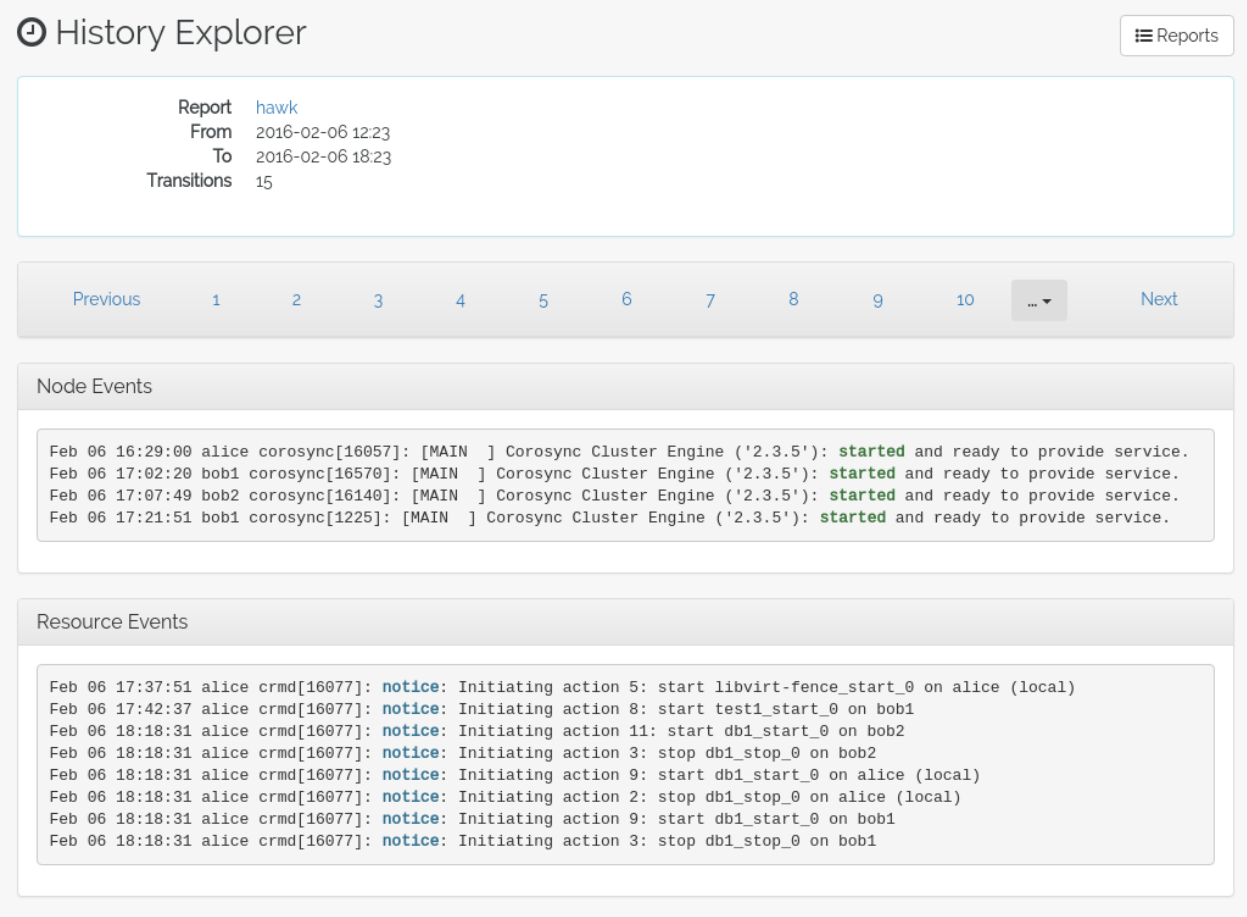
From the *History Explorer* pane, it is possible to create new history reports and upload a previously created report. A list of all created reports appear below, which options to view or download each report.



6.2 Viewing a Report

When viewing a report, the interface presents a list of *Transitions* that occurred during the time frame covered by the report. These Transitions are changes in the cluster state. Either they are triggered by changes in the configuration, or something happened like a resource failure or a node failure, and the cluster reacted to it in some way.

Below the transition list, there is a list of node and resource events. The history explorer analyses the given report and tries to sift out the key events so that finding the interesting section of the report is made a bit easier.



6.3 Video

The embedded video is somewhat outdated, and shows the history explorer as it looked in Hawk 1. However, it may be a good introduction to the basic functionality of the history explorer.

CHAPTER 7

Simulator

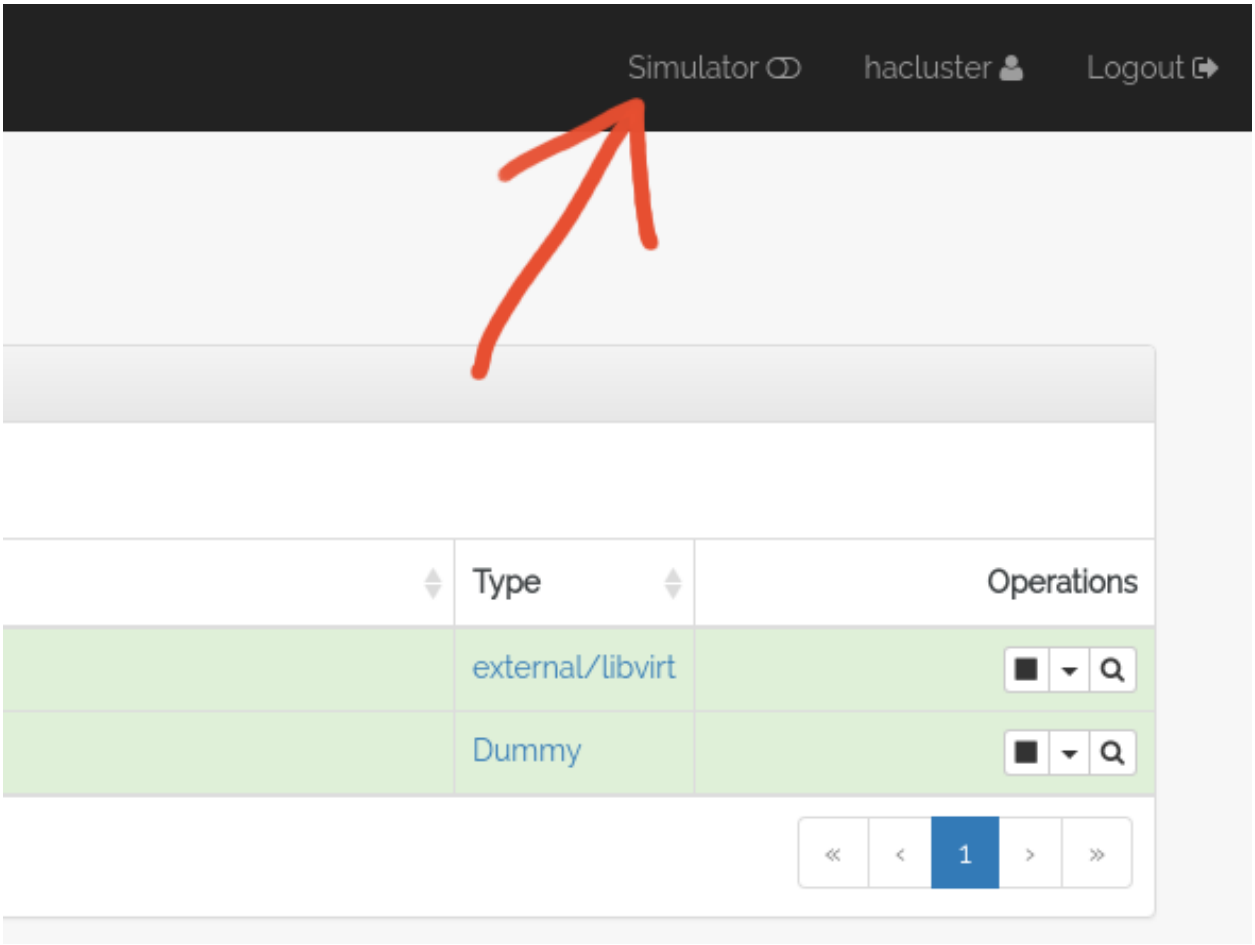
The *Simulator* in Hawk can be used to test changes to the cluster without actually changing the configuration. By enabling the simulator, resources and constraints can be edited, created or deleted freely, and the Status view will update instantly with the new expected state of the cluster. This makes it easy to test complex resource constraints, for example.

In this section, we will demonstrate that our fencing device works, first by testing the cluster resource using the Simulator, and second by actually triggering a fence event.

7.1 Fence Cluster Resource

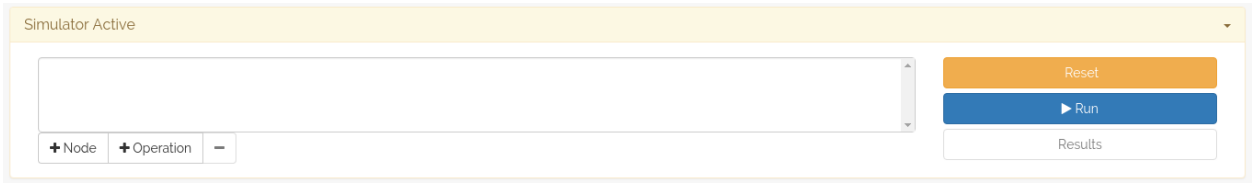
In *Configuring fencing* we configured a `external/libvirt` resource.

Using the simulator, we can test that the cluster configuration of this cluster resource is correct without actually triggering a reboot of any cluster node.

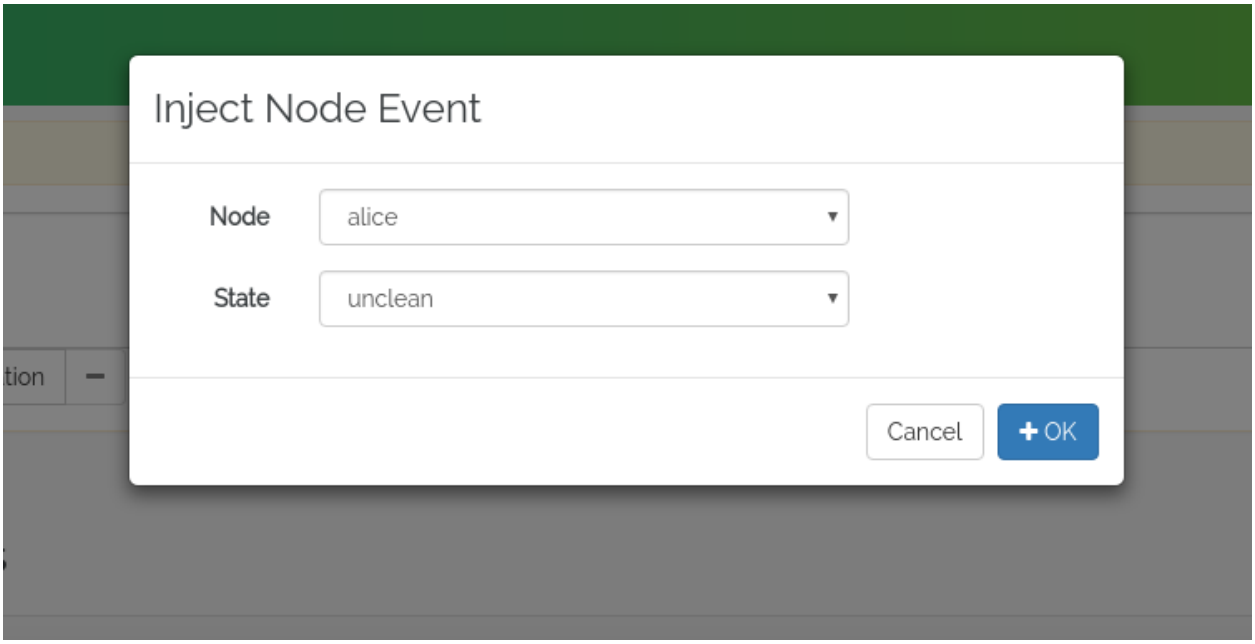


To do this, go to the *Status* view in Hawk and enable the Simulator using the toggle in the upper right corner of the window.

At the top of the main area of the Hawk application, a new panel should appear.

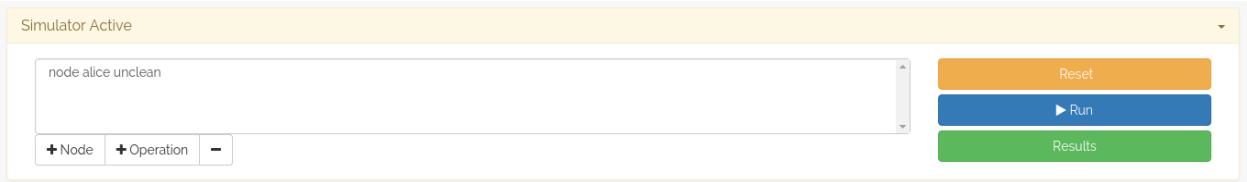


To simulate that the node `alice` experiences a failure that requires fencing, inject a *Node Event* using the **+ Node** button in the Simulator panel. This opens a dialog where you can choose which node and what event to simulate. In this case, we want to simulate an `unclean` event. `unclean` is Pacemaker code for a node that is in need of fencing.



The node event is added to the current simulation queue, in the middle of the simulator panel. To run the simulator, press the **Run** button on the right side of the panel.

The **Results** button will turn green, indicating that there are simulation results available to examine.

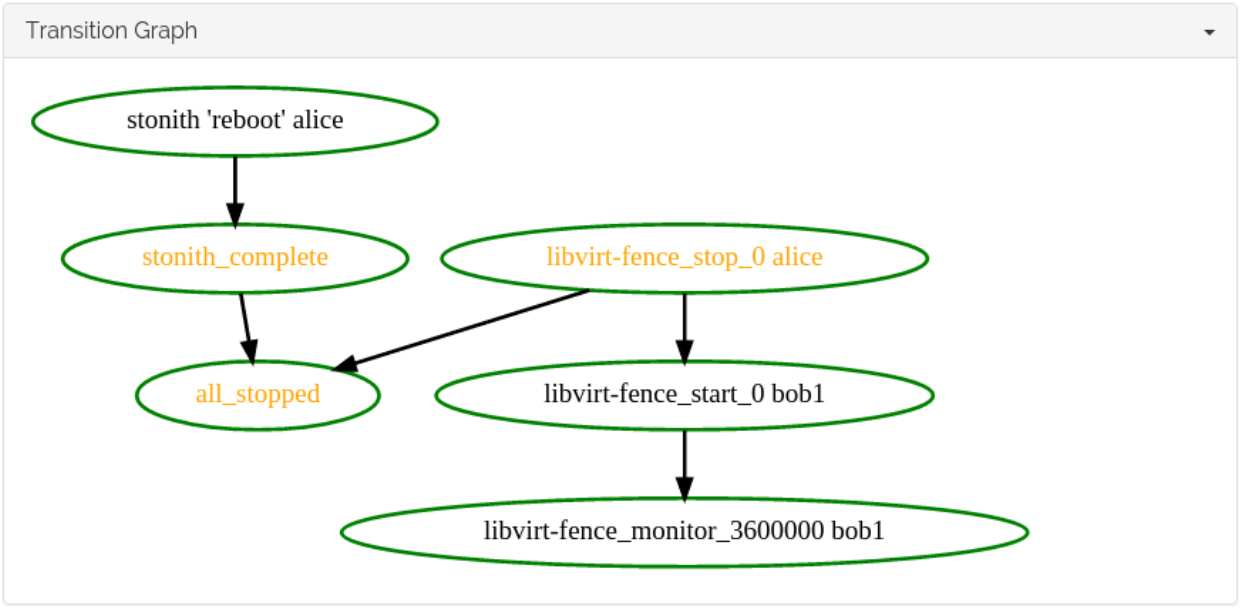


In the modal dialog that opens when the **Results** button is clicked, you can see a summary of the events that the cluster would initiate if this event occurred.

In this case, we can verify that the cluster would indeed attempt to fence the node `alice`, as seen on the second line in the transition list:

```
Executing cluster transition:
 * Pseudo action:   libvirt-fence_stop_0
 * Fencing alice (reboot)
 * Pseudo action:   stonith_complete
 * Pseudo action:   all_stopped
 * Resource action: libvirt-fence   start on bob1
 * Resource action: libvirt-fence   monitor=3600000 on bob1
```

The graph view can sometimes be confusing if there are a lot of actions being triggered, but this example serves as a nice introduction to reading transition graphs.



In the graph, we can see that the cluster reboots `alice`, and in parallel restarts the fencing device on another node (since it was running on `alice` before the event).

Exit the simulator using the the **Disable Simulator** toggle at the top right.

7.2 Fence Event

To trigger a fence event, we use the `virsh` command tool on the hypervisor machine.

Since we are going to make `alice` reboot, we should use Hawk from one of the other nodes. Go to https://localhost:7632/[1] to log in to `bob2`. The Hawk interface should look the same as on `alice`.

As `root`, run the following command:

```
$ virsh suspend hawk-guide_alice
```

At the same time, keep an eye on the Status view on `bob2`.

Suspending the VM has the same effect as a network outage would have, and the only recourse the cluster has is to fence the node which is no longer responding.

A sequence of events will occur in rapid succession:

1. The `libvirt-fence` resource which was running on `alice` will enter an unknown state (shown with a yellow question mark in the Status view).
2. The node `alice` may briefly appear to be in an `unclean` state, but will quickly move to `stopped` as the fence action completes.
3. `libvirt-fence` will start on one of the Bobs. It may briefly appear to be running both on `alice` and on one of the Bobs. Since `alice` has been fenced, Pacemaker knows that the resource is no longer running there. However, it is conservatively waiting until `alice` reappears before updating the status of the resource.
4. `alice` will reboot and turn green again.

[1] We are logging into `bob2` here rather than `bob1`, since we triggered a reboot of `bob1` while configuring fencing which unfortunately disrupted the port forwarding that Vagrant configured for us. We can still reach Hawk on `bob1` by connecting directly to its IP address: https://10.13.38.11:7630/

Fencing was successful.

CHAPTER 8

About this Guide

This is the user guide for Hawk, the High Availability Web Konsole.

The guide is published at Read the Docs as http://hawk-guide.readthedocs.org/ .

For more about Hawk, see https://github.com/ClusterLabs/hawk .

CHAPTER 9

Indices and tables

- genindex
- search