UpGuard Report Processor

Abstract

`upguard.rb` is a custom Puppet report processor that can be used to integrate Puppet, UpGuard and Jira to create server qualification and change validation tickets. The report processor is written in Ruby and adheres to Puppet report processor standards. The following sections describe how `upguard.rb` interacts with the PuppetDB (PDB) and with Jira to create the aforementioned ticket types. The latest version of `upguard.rb` can be found here.

Systems Involved

PuppetDB (PDB)

The PuppetDB collects data generated by Puppet. It enables advanced Puppet features like exported resources, and can be the foundation for other applications that use Puppet's data.

UpGuard

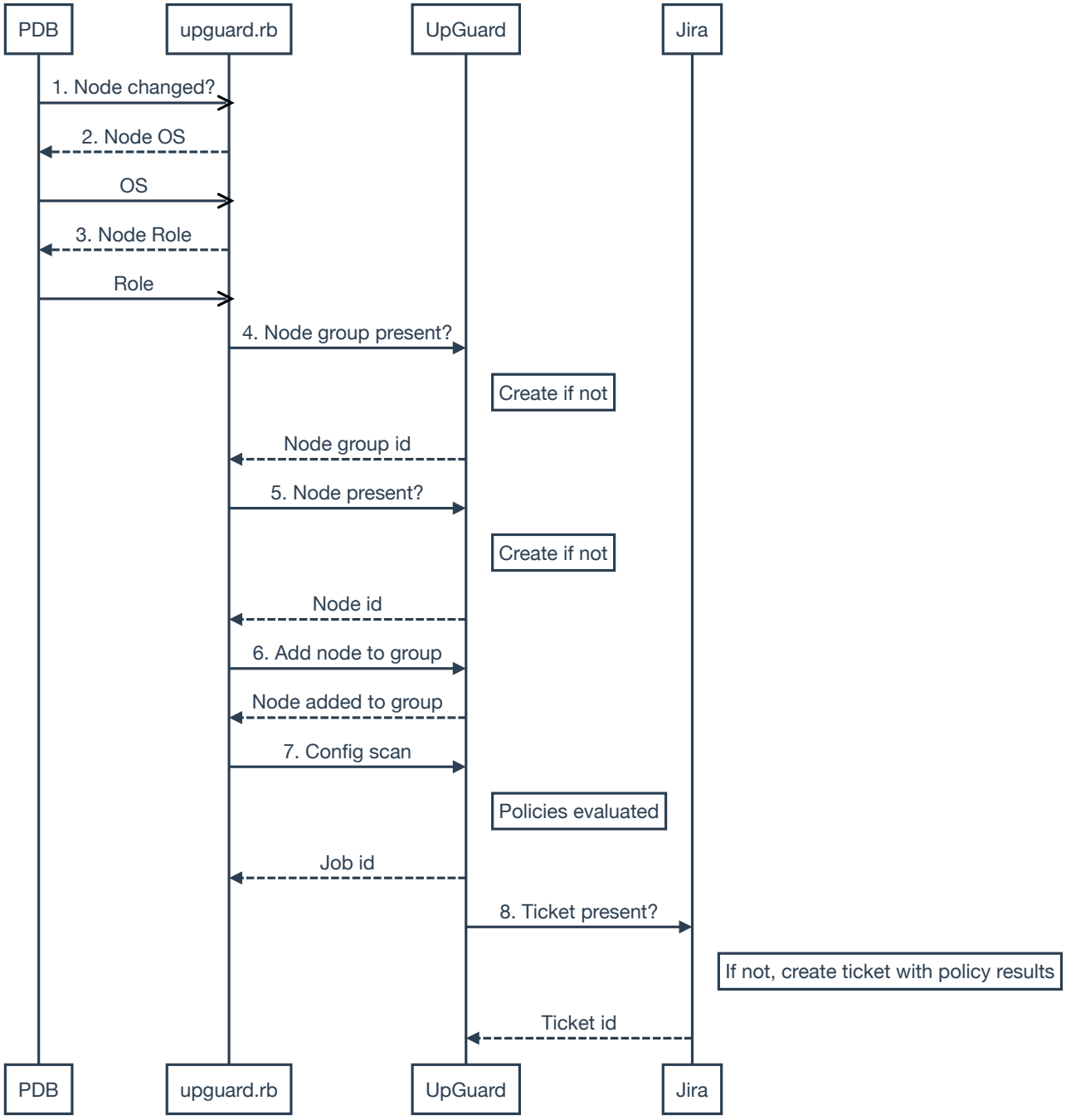
UpGuard collects server (node) configurations allowing for both API and visual based differencing as well as configuration compliance. UpGuard is deployed as a behind-the-firewall appliance.

Jira

Jira is a proprietary issue tracking product, developed by Atlassian. It provides bug tracking, issue tracking, and project management functions.

Sequence Diagram

The following diagram describes the sequence of messages between PDB, UpGuard and Jira as orchestrated by `upguard.rb`. Detailed message descriptions follow.



Messages

> **Note:**
>
> - All message response types are JSON formatted unless otherwise specified.
> - Messages with a `?` present a response decision for the recipient system.
> - Values encased in curly braces `#{}` are variables pointing to the current Puppet DB, UpGuard or Jira instance.

1. Node changed?

Any changes that Puppet makes to a node will trigger the execution of `upguard.rb`. This report is executed after a Puppet agent run has completed and ensures that Puppet and UpGuard nodes and roles remain synchronized.

Type	Method	Interface	Response
Asynchronous	Internal	`Puppet::Reports.status`	Changed

2. Node OS

An additional call to the PuppetDB is required to determine the nodes operating system as this is not a variable that is exposed through the `Puppet::Reports` class. Nodes created in UpGuard require the operating system field to be set.

Type	Method	Interface	Response
Synchronous	GET	`#{PUPPETDB_URL}/pdb/query/v4/facts/operatingsystem`	OS type

3. Node Role

As with determining the nodes operating system, an additional call to the PuppetDB is required to determine the nodes role (via trusted facts).

Type	Method	Interface	Response
Synchronous	GET	`#{PUPPETDB_URL}/pdb/query/v4/nodes/#{node_ip_hostname}/facts`	Node role

4. Node group present?

Roles in Puppet are synonymous to node groups in UpGuard. Any roles that exist in Puppet should exist as a node group in UpGuard. An attempt is made to add the node group to UpGuard, if the node group already exists, its `id` will be returned.

Type	Method	Interface	Response
Synchronous	POST	`#{UPGUARD_URL}/api/v2/node_groups`	Node group id

5. Node present?

Any nodes that are being changed by Puppet will need to be monitored by UpGuard. With the node's operating system and role details determined above the node can be created in UpGuard, keeping Puppet and UpGuard nodes synchronized. If the node already exists in UpGuard, its `id` will be returned.

Type	Method	Interface	Response
Synchronous	POST	`#{UPGUARD_UR}/api/v2/nodes`	Node id

6. Add node to group

Attaching the node to a node group in UpGuard will allow it to automatically inherit the relevent scan options and policies attached to the node group. Subsequent configuration scans of the node will allow policy results to be determined. This step is the last in synchronizing a Puppet node (with a role) to an UpGuard node (with one or many node groups).

Type	Method	Interface	Response
Synchronous	POST	`#{UPGUARD_URL}/api/v2/node_groups/#{node_group_id}/add_node.json?node_id=#{node_id}`	Node id

7. Config scan

Configuration scans allow for a nodes known configuration state to be tracked over time. Once a node's state is known to UpGuard after its first successful configuration scan, policy results are determined automatically. Policies in UpGuard allow for users to determine a nodes desired state and can be written in advance of the node or role existing in Puppet allowing users to write tests in advance or in conjunction to Puppet roles.

Type	Method	Interface	Response
Synchronous	POST	`#{UPGUARD_URL}/api/v2/nodes/#{node_id}/start_scan.json`	Job id

8. Ticket present?

Policy success and failure results, on a per node basis, are captured through server qualification tickets in Jira. If a ticket with the same title already exists for a node, another will not be created. A server qualification ticket with policy failure results can enter into a Jira workflow where it can be assigned to a user or group for remediation. In this case, the Puppet role would likely be reviewed and modified rather than the cited node. A new node with the updated Puppet role can then be supplied for subsequent retesting.

Type	Method	Interface	Response
Synchronous	POST	`#{JIRA_URL}/rest/api/2/issue`	Ticket id