Take Control of Datacenter Infrastructure

Uniting the Governance of a Single System of Record with Powerful Configuration Automation Tools
Enterprises that have implemented strong change and configuration management processes typically have the highest levels of availability. The last mile for these processes is the actual configuration of the infrastructure itself. Yet infrastructure configuration has largely remained a manual, tedious, and error-prone activity. Legacy software products have tried to address this challenge using an approach grounded in the enforcement of static policies. However, they lack the operational agility required to manage a dynamic infrastructure. To address this challenge, a new breed of infrastructure automation products has emerged that transforms infrastructure into IT-accessible code. Yet as powerful as these products are, they lack the level of transparency and governance required by modern enterprise IT. This white paper describes a powerful, unique, and differentiating single system of record approach to infrastructure configuration automation that leverages open source automation software from Puppet Labs.

**Understanding Infrastructure**

The enterprise is a complex organization of people and technologies. Enterprise architecture provides the essential business logic for business processes as well as the IT infrastructure and IT processes that support an operating model for the business itself. Most enterprises have become almost entirely dependent on IT to be able to deliver goods and services to customers.

A model for IT infrastructure components is offered in figure 1, and definitions of the components are included in table 1 below.

![Figure 1: IT infrastructure components model](image-url)
### Configuration artifacts

Configuration artifacts are bundles of code and data that define the configuration of the software and infrastructure services hosted on nodes.

<table>
<thead>
<tr>
<th>Component</th>
<th>Definition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Artifact</td>
<td>Data and code used to manage the configuration of infrastructure services and software</td>
<td>Puppet Modules</td>
</tr>
<tr>
<td>Infrastructure Device</td>
<td>Hardware that can be assigned software realized by configuration artifacts</td>
<td>Rack-mounted blade server</td>
</tr>
<tr>
<td>Node</td>
<td>A computational resource within a device that houses infrastructure services whose configuration is defined by configuration artifacts</td>
<td>Amazon EC2 instance</td>
</tr>
<tr>
<td>Software</td>
<td>A set of machine-readable instructions that directs the behavior of a node to perform a certain function</td>
<td>MySQL software</td>
</tr>
<tr>
<td>Infrastructure Service</td>
<td>Functionality with a well-defined interface hosted on one or more nodes</td>
<td>Apache Tomcat service</td>
</tr>
<tr>
<td>Network</td>
<td>The physical infrastructure used by two or more devices to communicate</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>Network Path</td>
<td>The logical connection on the network used by two or more nodes on the network to communicate with one another</td>
<td>Network route between two devices</td>
</tr>
</tbody>
</table>

**Table 1: Definitions of IT infrastructure components**

One key concept in this model is the distinction between devices and nodes. Logical partitioning of hardware is a long-standing capability of IT infrastructure. The rapid proliferation of cloud technologies has increased the importance of this distinction.

Another key concept is the role of configuration artifacts. Configuration artifacts are bundles of code and data that define the configuration of the software and infrastructure services hosted on nodes. A practical example of a configuration artifact is a Puppet Module.

Puppet Modules are self-contained bundles of Puppet code and data. Puppet admins develop their own modules or download modules already created by the Puppet community from Puppet Forge. Modules essentially define the configuration artifacts (in the form of Puppet code) that Puppet can use to manage the state and configuration of software or infrastructure services.

Separating configuration artifacts from infrastructure components means IT admins can spend less time thinking about nodes (the obsession of static rules-based, legacy configuration automation tools) and think more about the definition of the desired state of software and infrastructure services. Specifically, they can focus on the development of configuration artifacts, such as Puppet code and data, that drive the behavior of the infrastructure.

This essential transformation in thinking brings with it the opportunity to re-evaluate the processes that are used to manage IT operations. Infrastructure and code begin to merge and with this transformation comes a tighter collaboration between software development and IT operations – infrastructure as code.
Node Definitions

As new operating models emerge or existing ones evolve, the IT infrastructure must adapt to support them. This can be an enormous challenge, especially for rapidly evolving businesses that have complex infrastructures. Managing such an environment through the enforcement of static policies, not to mention enforcing and updating them, can be an enormously complex task.

Consider a traditional enterprise change management process. A change needs to be described which generally involves understanding the state of the resource being updated. Once the change request is submitted, the change process needs to be followed. Once approved, the change needs to be implemented. While there may be little or no automation, the process is well-governed by change management.

In contrast, Puppet Labs provides IT administrators the ability to manage a dynamic infrastructure via Puppet's declarative language. The code IT administrators develop for Puppet is extremely powerful. Puppet can easily automate application deployments and other changes. A relatively simple change in rules on the Puppet Master could impact hundreds or thousands of infrastructure elements with minimal opportunity for change management.

With this great power comes the great responsibility of integrating Puppet code into established ITSM processes. But configuration artifacts are nothing more than code and data, often making them challenging to integrate into the ITSM processes required to govern enterprise IT.

A ServiceNow node definition (illustrated in figure 2) is a collection of configuration artifacts that can be applied to a configuration management database (CMDB) configuration item (CI). Simply put, a node definition is the collection of configuration artifacts that defines the desired state of the CI.

In the context of Puppet, a node definition would be a collection of Puppet Classes discovered from a collection of Puppet Modules. For example, an IT admin might develop Puppet Modules for infrastructure services. Additional modules developed by the Puppet community might also be downloaded from Puppet Forge. Taken as a whole, the manifests contained in these modules define the code and data that can be used by Puppet to configure the infrastructure.

Figure 2: ServiceNow node definition

The instantiation of a node definition does not alter the behavior of the node itself – it is simply a definition. The application of the node definition to the CI establishes the desired state of the CI based on the code contained in its configuration artifacts. The process used by ServiceNow to manage node definitions is illustrated in figure 3. Discovered Puppet code is used by node definitions to define the desired state of CIs.
Once node definitions are applied to configuration items, one can use the CMDB as the single system of record for the desired state of these infrastructure components.

Figure 3: ServiceNow management of node definitions

Once node definitions are applied to CIs, one can use the CMDB as the single system of record for the desired state of these infrastructure components. The process for this is illustrated in figure 4. A node in the IT enterprise with a Puppet Agent connects to the Puppet Master where it looks to determine the configuration artifacts that should be applied to it. The Puppet Master looks up the CI in the ServiceNow CMDB (via its fully qualified domain name and external node classification) and determines the configuration artifacts that should be applied. Essentially the Puppet Classes and data define the CI's desired state.

Configuration Item

Node Definition B
- Configuration Artifact
- Configuration Artifact
- Configuration Artifact

Device

Node
- Infrastructure Services
- Software
- Puppet Client
- Configuration Artifact

Node Classifier
- Puppet Master

Figure 4: Single system of record for the desired state of infrastructure components

Node definitions in ServiceNow unite the power of Puppet with the governance of the ServiceNow CMDB. They bridge the gap between configuration artifacts and ITSM processes such as change management.

Change Management

Node definitions could potentially be applied to a large number of systems. Once a node definition is updated, the change would be rapidly picked up and implemented by Puppet.

Consider a scenario where an emergency security update needs to be deployed. Updating a deployed node definition to apply this security update would almost immediately begin its deployment. This would probably not be permitted in enterprise
ServiceNow inserts change management as a governance layer into the process of changing node definitions.

IT without being first reviewed and approved in a formal change management process. Perhaps the proposed update needs to be tested before it is approved for deployment into production. While the power of Puppet automation is remarkable, this example serves to illustrate the importance of change management as a governance layer.

ServiceNow manages this by inserting change management into the process of changing node definitions. An IT admin can check-out a node definition and create a draft for editing outside of production. Changes made to a draft do not change the node definition applied to CIs in the CMDB. Once the draft is ready, it can be submitted for change management approval by requesting the update be published. This three-step process is illustrated in figure 5.

![Figure 5: Node definition change management process](image)

As an illustration of this process, refer to figure 6. In this scenario, an IT admin wants to add a configuration artifact from a node definition. He checks-out the node definition and makes the desired change. Once complete, he publishes the new node definition via the change management process. Once approved, the node definition is updated and Puppet begins applying the new configuration which includes the new configuration artifact.

![Figure 6: Adding a configuration artifact to a node definition](image)
upgrade or to repurpose a node for a different infrastructure function. Once again, this change will impact the configuration of the server, so the change management process needs to be followed. Until the change is approved, the new node definition will not be applied. Once approved, the CI can be updated with the new node definition and Puppet can proceed with the application of the appropriate configuration artifacts.

Figure 7: Changing a node definition applied to a CI

Cloud Provisioning
A typical approach to service catalog-based cloud provisioning involves discovery of virtual machine (VM) templates and the publication of these templates as orderable items in the service catalog. When a self-service user orders a VM, the resource is cloned from a template via workflow-driven automation. Management of VMs provisioned in such a fashion consists of general resource accounting and lease management. These techniques typically excel at addressing challenges such as VM sprawl, where poor accounting of virtual resources leads to diminished returns on investments in cloud resources.

However, this approach only manages the VM as a container and does not address the configuration of the resources hosted on the VM itself. Also, nothing is done to manage the configuration of the VM for the duration of the lease. Consider the scenario where a VM template behind a service catalog item has to be patched. The VM template could be updated, and all future VMs would receive the security patch. But what about the potentially hundreds or thousands of VMs already provisioned and under lease management against this service catalog item? They also require the security patch. However, because they have already been provisioned their configuration cannot easily be updated. There is no way to maintain the VMs in a desired state once they have been provisioned.

In addition, there is the overhead of managing VM templates mapped 1-to-1 to service catalog items as illustrated in figure 8. If the previously mentioned security patch had to be applied to several VMs in the service catalog, then each VM template would have to be re-built to reflect the new desired state. For large service catalogs, this would be a lot of work. This is a consequence of provisioning VMs without the ability to manage them to a desired state.
This approach is especially beneficial to large service catalog deployments where it would be easy to be overwhelmed by VM templates were it not for the application of configuration automation node definitions.

Fortunately, node definitions provide a mechanism to address these challenges. Rather than creating many VM templates that map 1-to-1 to service catalog items, all VMs are created from a single base VM that contains just enough of the base operating system to boot up, attach it to the network, and launch the agent for the configuration automation system (for example, the Puppet agent). Fulfillment of a service catalog item then involves cloning this base VM and assigning the node definition that matches the requested service catalog item. When the VM boots up, the configuration automation system applies the desired configuration and the VM is configured and handed over to the self-service user. This approach is illustrated in figure 9.

Only one base VM template is required for a collection of service catalog items. Hence there are potentially far fewer templates that need to be managed and fewer conditions that would require the base VM template to be updated. This approach is especially beneficial to large service catalog deployments where it would be easy to be overwhelmed by VM templates were it not for the application of configuration automation node definitions.

Once provisioned, the VM is managed to this desired state. It can be updated as needed by simply adjusting its node definition. To follow the earlier example of a security patch that needs to be deployed, adding a configuration artifact to the node definition for the security patch will cause all future VM requests to be provisioned with the patch. All VMs that have already been provisioned will likewise be updated to match the desired state of the VM. This means the configuration of the VM can be managed throughout the duration of the VM lease.

Self-service Change
The self-service concept starts with the service catalog but does not end there. For example, there should be no need for self-service users to make requests of IT to perform basic management tasks such as stopping, starting, and terminating VMs. Self-service actions from a management dashboard should easily accomplish such tasks.
Likewise, self-service users can be empowered to re-configure their virtual resources via the service catalog by simply selecting a node they have provisioned and requesting a different node definition. While change management processes must be followed, the fulfillment of such requests is as simple as changing the node definition associated with the CI of the VM being reconfigured. Puppet will update the node to match the new desired state.

**Metrics of Success**

The list of metrics listed below is not intended to be exhaustive. What is more important than an impressive list of metrics is the reason they are measured and acceptance of these measurements by the business. IT is notorious for making all sorts of measurements for its own consumption, few of which have any value (let alone meaning) to a partner in the business who cares about business value.

<table>
<thead>
<tr>
<th>Category</th>
<th>Metric</th>
</tr>
</thead>
</table>
| Agility  | • Time from receipt of configuration request to handoff of a resource  
           • Time to integrate a standard configuration into the infrastructure |
| Efficiency | • Percentage of infrastructure whose configuration has been modeled  
                • Average number of systems per member of operations staff  
                • Ratio of number of VM templates to VMs offered on Service Catalog |
| Flexibility | • Time from receipt of request to reconfiguration of infrastructure resource |
| Reliability | • Time to rebuild a failed component from a standardized configuration  
                     • Percentage of VMs configured without incident  
                     • Percentage of configurations automated and tested |

**Table 2: Key metrics of configuration automation success**

Make commitments and proactively demonstrate accountability using reports driven by key process metrics. Reporting, if appropriately implemented, remains an outstanding way to establish customer intimacy. Understand the metrics your customer most values that reflect the agility, efficiency, flexibility, and reliability of IT services. Add metrics and categories as needed to create the feedback loop with customers required to create alignment and foster customer intimacy. The reports should directly relate to the operational and governance processes.

Think critically about this data. Why are you measuring a given metric? How is the metric evaluated, and can we automate its collection? Can we formalize a given metric in a Service Level Agreement? What do we do when the value falls outside of expectations? What do we do when the results exceed expectations? How do the metrics we are collecting map to operational support process? Are there governance key risk indicators (KRIIs) or key control indicators (KCIIs) associated with one or more metrics?
What about so-called intangible or indirect metrics that are difficult or impossible to measure? Acknowledge they exist. Even if they cannot be measured, it is important to document what these metrics are and agree they are intangible. Establish a clear understanding of why they are not being evaluated. Even if the only data that can be collected for these metrics is anecdotal, there is still value in collecting it. Challenge teams to find ways to identify ways to measure them. Working with customers might bring opportunities to the surface that would have otherwise gone unexplored.

The Next Step
Open source configuration automation products like Puppet transform infrastructure into IT-accessible code. They provide IT with a language of their own that they can use to solve challenging and dynamic infrastructure problems. Using this language, IT teams can manage infrastructure projects using iterative design processes such as Agile as part of a mature software development lifecycle. This is the next frontier for IT – infrastructure as code with a merged set of DevOps processes working within the framework of strong ITSM governance.