Matthew Sacks
GlassCode Inc.

Datacenter in a Wiki

By automating and better leveraging documentation, you improve your infrastructure, process, and code.

The primary concept of the solution for this problem marrying the engineering systems and the simplicity of human expression as a way to build and maintain systems.

Problem(s)

We were producing many server farms for many different projects with different configurations, and keeping track of the different documents, or Infrastructure Design Documents, which outline how a given Web stack would be built (it's server assets, network configuration, and applications installed on the server). These documents would vary in how they were stored and formatted amongst groups; if you were in the network team you stored all of the information in a spreadsheet, if you were on the applications team, you stored in in an unformatted Wiki page.

The melange of information passed through many different hands of many different teams in different formats, and many errors were introduced.

Much time was wasted in the building of a server farm for a given software product, in this case a Web application stack, and people simply didn't have time to continue to update documentation, even if it is introducing errors into daily operations as a result of misinformation. By leveraging automation and standardizing common documents, the process of improving infrastructure and code becomes more effective, and process is improved as a result. In this presentation, I use the specific example of building many server farms using documentation processes as an example, but the principles can be used to address many other similar problems.

Other Similar Problems

Stale Documentation Problem:

Aside from the lack of automation and standardization in our documentation processes as they relate to operations, many operations groups experience problems with documentation becoming unmaintainable or getting out of date quickly.

Stale Documentation Solution:

One way of managing stale documentation is to add a timer to every document produced. If a document gets beyond a certain age, then it should alert a document administrator or technical writer that the page needs to be updated, or deleted if it is too old.

No time to update documentation problem:

Many engineers in operations and development alike claim that they simply do not have time to update documentation. Updating documentation is just as important as updating code, because through expressing your application and process designs in a written or verbal way, brings to light new improvements or issues that need to be addressed that otherwise would not be able to be rationalized or analyzed by just looking or thinking about the code or process.

Not documenting code, applications or process contributes to making legacy systems, and legacy systems can be a huge business liability if no one knows how to manage them.

No time to update documentation solution:

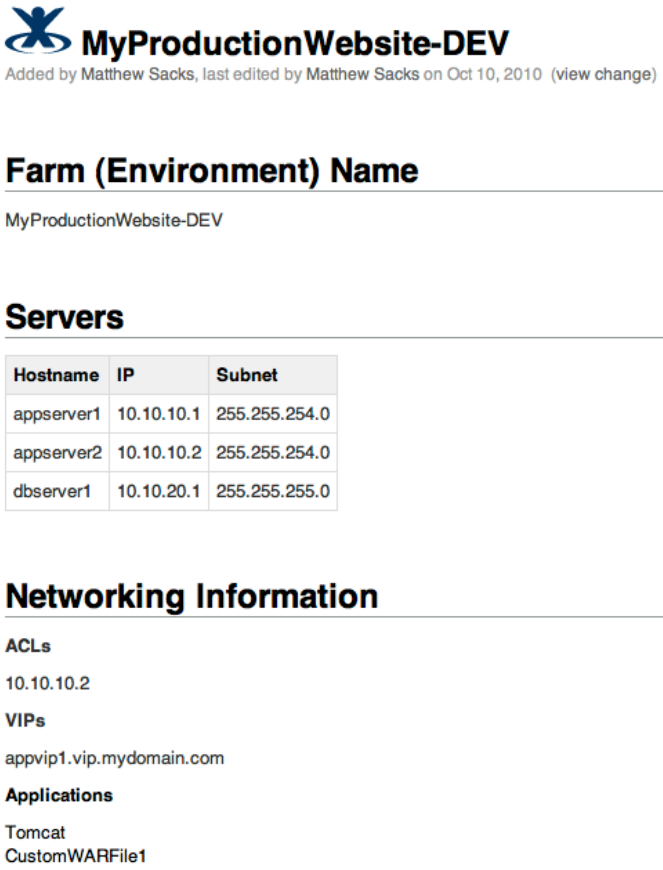
Make it a necessary part of the software development life cycle, for operations employ templates to automate the documentation of process and infrastructure designs by integrating documentation systems with configuration management and provisioning systems, so that documentation now becomes functional rather than just words on a page.

Also, by providing feedback mechanisms such as rating or comment systems and actually using them, the need to update a particular document becomes more apparent. Not all documentation needs to be updated, so by soliciting the users of a Documentation system or Wiki, one can become aware of what needs to be updated most.

Standard Template Solution

To address this problem, a template was created in Confluence to consolidate all of the information required to build out a particular server farm including all of the applications, dependencies, hostnames, IP addresses and networks to run an entire Web site, for example.

Creating a standardized Infrastructure Design Document helped reduce the number of errors and confusion around how these server farms should be built, but they were not fool proof, and a human still had to translate the documentation and manually input it into a configuration management and provisioning engine to provision and configure the servers, applications, and configurations.



MyProductionWebsite-DEV

Added by Matthew Sacks, last edited by Matthew Sacks on Oct 10, 2010 (view change)

Farm (Environment) Name

MyProductionWebsite-DEV

Servers

Hostname	IP	Subnet
appserver1	10.10.10.1	255.255.254.0
appserver2	10.10.10.2	255.255.254.0
dbserver1	10.10.20.1	255.255.255.0

Networking Information

ACLs

10.10.10.2

VIPs

appvip1.vip.mydomain.com

Applications

Tomcat
CustomWARFile1

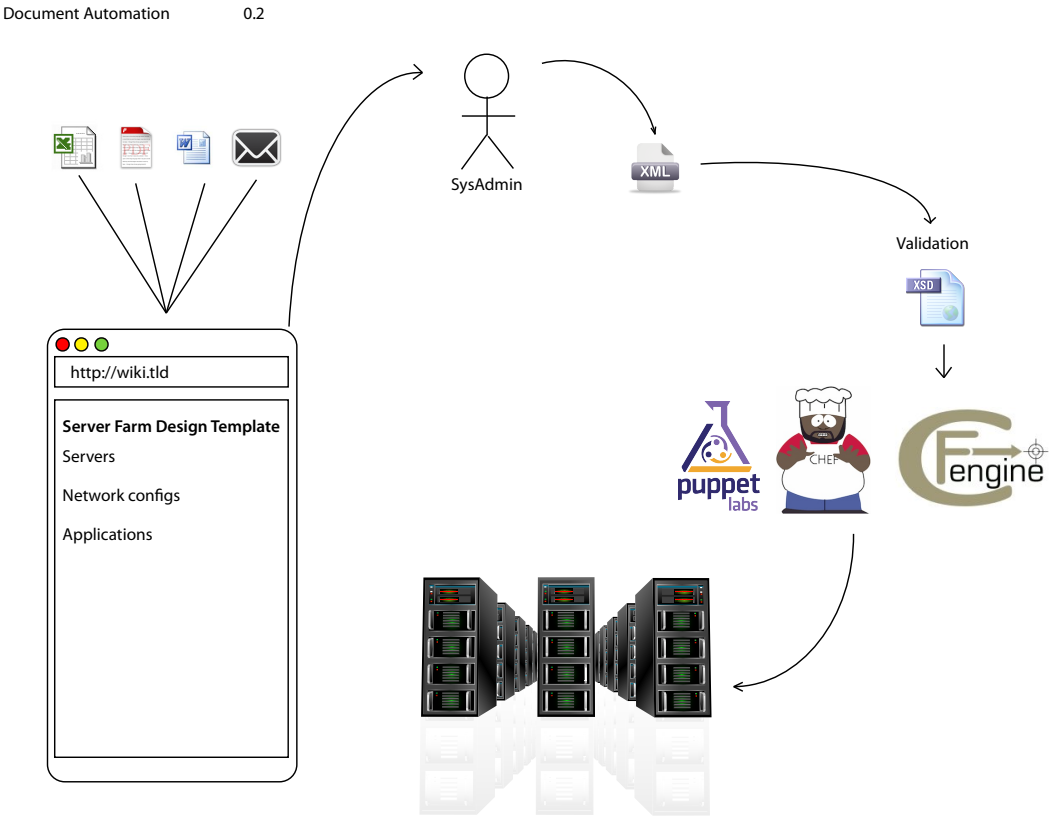
An early example of an Infrastructure Design Document

The Current Solution

There still a great deal of human error introduced in the process because of the lack of validation. The Wiki did not have the ability to validate a given document against a pre-defined structure, such as an XML Schema Document, for example, so we built a utility in Python to validate an XML document against an XSD to ensure that the Infrastructure Design Document was inputted properly into the Configuration Management engine.

The shortcoming of this was that we did not have any integration between this XML file and the original Wiki page which was being consumed by all of the groups, so we lost a great degree of automation in that and the XML document had to be translated based off of a Wiki page.

One of the great problems was that there were many different groups, again, who prefer their own format and their own way of doing things, which made it difficult to get everyone on the same page. In order to do this we had to integrate bit by bit within each team using a common format using a standardized template within the Confluence wiki.



Future Solutions

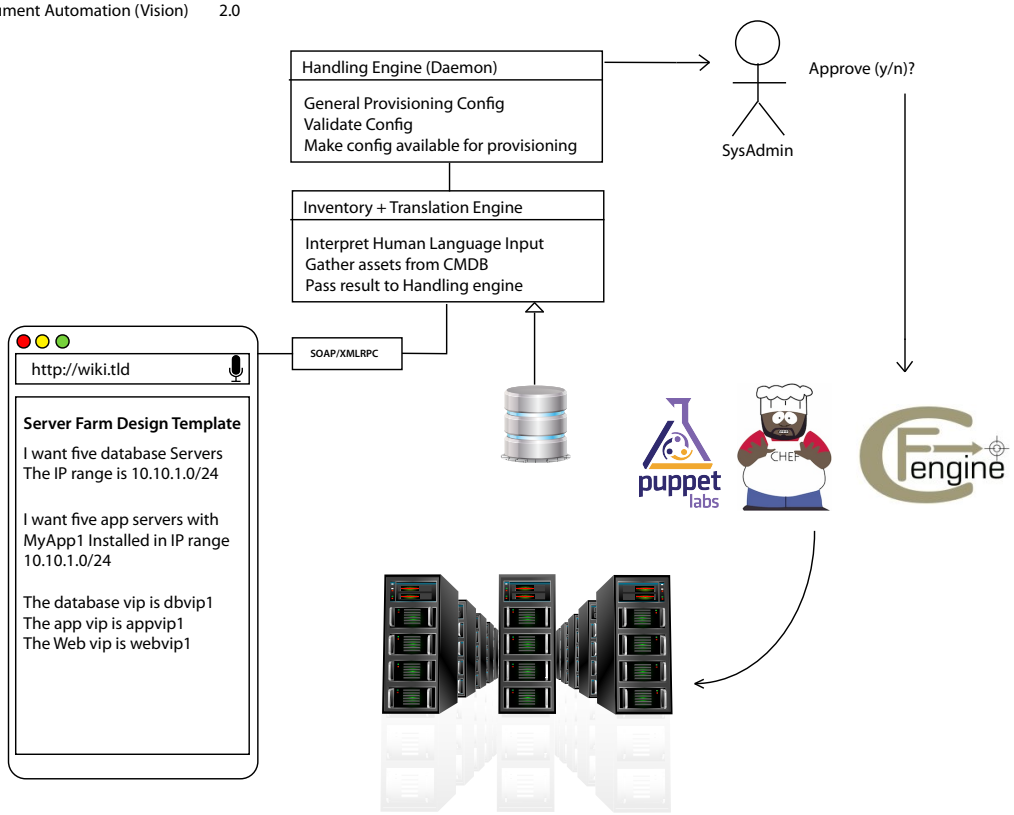
1.0

Although it is still in development, there are still many errors in the current design. Manual intervention by a system administrator is still needed to perform the translation of a Infrastructure Design Document into XML, and then the XML must be manually validated. In future solutions, an application will read the Infrastructure Design Document via SOAP or XMLRPC into a handling engine, which generates the XML and validates and automatically feeds it into the provisioning system. This will still require a system administrator to approve the buildout, because there still needs to be a human being moderating build requests, otherwise things could get out of hand very quickly.

2.0

Taking this design a step further, a translation engine can be added which takes design arguments in plain English, similar to the Cucumber project [1] which then gathers assets from a configuration management database (CMDB) which has a pre-defined pool of assets (virtual servers, VIPs, network addresses).

In this design, a person may simply say expressively in a Infrastructure Design Document they would like “5 application servers in network range 10.10.1.0/24 with a VIP name of app1.mydomain.tld”, for example. The advantage is that the system administrator no longer has to keep track of assets to be provisioned, anyone can request a new server farm (pending final approval from the system administrator), and all of the components and actions necessary to build a server farm are now managed through an expressive, standard Document.



Version 2.0 of the Infrastructure Design Template Provisioning Workflow

Conclusions

By automating and standardizing documents in a Wiki or other documentation system, and integrating it with other automation tools such as configuration management and provisioning systems, one can create a central point for creating complex server farms and keeping inventory of their relevance using such systems. By using an expressive format for creating infrastructure such as a Document Template, anyone can create server farms or application environments and mitigate the confusion introduced by the collaboration of multiple cross-functional operations and development teams in building a complex Web or application stack.

By employing similar methods to the creation of infrastructure by Documentation standardization and automation, such as adding timers to documents, and standardizing the template of a given application or process, one gains the features and benefits of keeping code and process updated, and finding design flaws or improvements which can be made in an IT organization by using an expressive process like documenting.

Giving automation and standardized template facilities to the end-user, and integrating it into common operational workflows, one can take the reluctance to create and maintain documentation, which ultimately benefits all aspects of an IT organization or group.