POCs And Reference Implementations

From HUIT Architecture Advisory Group

Contents

[hide]

Proofs of Concept and Reference Implementations

On this page you will find information about POCs and Reference Implementations. In most cases, the POCs will evolve to be reference implementations. These reference implementations can be used to help engineers refactor existing projects, or act as templates for new projects.

September 1, 2013 Proof Of Concept Demonstration

This POC will show key points and viability of some of the elements of the evolving HUIT enterprise architecture. Two use cases have been selected for this POC. A three tiered architecture that has been crafted based on the requirements anticipated for a new generation LME, based on a familiar three tiered application architecture. Also selected is a HUIT Unix Application Development environment. These architectural patterns, using AWS technology, will be used to illustrate key objectives of the enterprise architecture effort. Specific objectives for this POC include:

1. Simple, reproducible POCs that people can (with a minimum of change assuming they have AWS accounts) run themselves and experiment and learn from. This will be a model for other outputs from the group - that is that and code developed will be freely available.
2. Demonstrate a framework for reliable applications that reduces single points of failure using key building blocks and design patterns.

3. Illustrate a less complex deployment architecture that is not only reliable, but scalable and at the same time provides security properties designed to meet requirements of the majority of our applications. An assumption for this POC is that infrastructure services that we rely on such as DNS, Routing, etc are reliable and sufficient at least for the POC. It is also understood that variations of this basic pattern might be need in those cases where access to internal resources is required. In cases where this environment needs to integrate with non-EC2 resources, a different approach that has more traditional granularity based on network addressing might be required.
4. Demonstrate continued uptime in the event of a single element, stack or site wide failure.
5. Illustrate speed of scaling/adding new deployments with little human interaction in a very cost-effective fashion.
6. Illustrate a new model for development that incorporates more autonomous project teams creating development and test environments from templates. This will improve productivity while at the same time reducing both labor and equipment cost.
7. Illustrate a way to reduce operational costs though the use of architectures that significantly reduce the need for more expensive traditional DR and backup approaches - though not all facilities are implemented in this POC.
8. Show how developers can cost effectively have multiple systems, spin them up on their own and create complete test environments that closely resemble the production configuration, the lack of which is a key risk element when we upgrade systems.
9. Illustrate a cross-organizational ad-hoc development team's effectiveness.
10. Demonstration of an integrated provisioning process that creates and configures everything from a virtual data center to a configured and running project collaboration site. See Deployment Activities in a Virtual Environment for more details. **NOTE WELL: Outside the scope of this POC is the detailed analysis and recommendations for a default set of technologies, procedures, and systems to be used to configure and manage these architectures when placed in operation.** That is an enterprise architecture project planned when resources and time permits. For additional details see: Integrated Tools and Operations for Configuration in the Future Projects section of this site.
11. Provide open access to the code used in the POC's to allow other groups to learn, experiment and provide feedback.

It has yet to be determined how far we will get on all these objectives. That will be determined by available resources and time.

Three-tier Application

This POC is based on the generic 3-tier application that used a potential future LME to drive its requirements. It is implemented using Amazon AWS technology. A key objective of the POC is to vet the technology as well as key design pattern in our enterprise architecture. As with other use cases, it is possible to meet the requirements of this use case with other technology. From this POC we will learn how easy it is to deploy this common but complex use case and have a better way of estimating operational costs using this approach. There are a number of technologies/features that may be of interest in the reference implementation for this use case and the software that realizes the new LME, that are beyond the scope of this effort due to time/resource constraints. They include:

1. Use of AWS resources across regions to support DR and extended back up functions.
2. User of Elastic Data Pipeline for workflows - including cross region.
3. Usage of DynamoDB for storage of session data or any other purpose.
4. Using a 'pilot light' type of approach in another region in the event of a failure so that traffic could immediately be diverted to the new location (until that point there is not production traffic going to that region) that would be scaled up from the systems to meet full production demand. This POC does not demonstrate any of the data backup/synchronization features required to keep a pilot light region ready.
5. Separate backup using Glacier or other technologies.
6. Operational issues such as providing for different environments (i.e., development, test, production).

Technologies for configuration and management (if any) are intended solely to stand up the POC and neither the technologies nor the scripts that drive them should be thought of as other than drafts certain to either be significantly changed or discarded. This POC will implement and document the following in support of the identified use case.

1. CloudFormation Scripts drive the AWS environment - this includes setting up at least two availability zones.
2. A stand alone VPC which may not have any connection to the Harvard environment. It will not require nor use a VPN. Future reference implementations will likely add these facilities.
3. Autoscaling groups/facilities for front end and business logic servers.
4. Security Groups will be created and all systems will be in at least one.- Even the ELB that sits in front of the two availability zones is in a security group.
5. Creation of instances for each of the system types (e.g., front end server and business logic server) for each of the availability zones. Autoscaling will cause

the creation of additional instances. These instances will be based on a basic Amazon Linux AMI to which we will apply customizations.

6. CloudInit capabilities will be used to bootstrap Puppet.
7. Puppet will be used to configure the EC2 instances once created.
8. If time permits, we will create a Puppet Master in a separate autoscale group. Otherwise local Puppet agents on each instance will be 'feed' information required.
9. For the POC, the sequence of change management for configuration changes will be to apply changes in Git (stored using S3). After storage in Git a Post-Receive hook will be used. RPMs might be used in a reference implementation.
10. Basic CloudWatch monitoring/verification.
11. A multi area zone RDS environment will be created to support the database in the example application (MediaWiki)
12. File storage will use S3 Our MediaWiki example requires a file plug in so that it can use S3 facilities.

Unix Application Development Environment

This POC is also implemented with AWS technology but could be implemented with other technologies. It is based on the Unix application development environment use case. **For this initial POC, the following areas are excluded,** though they would have to be provided in a complete reference implementation:

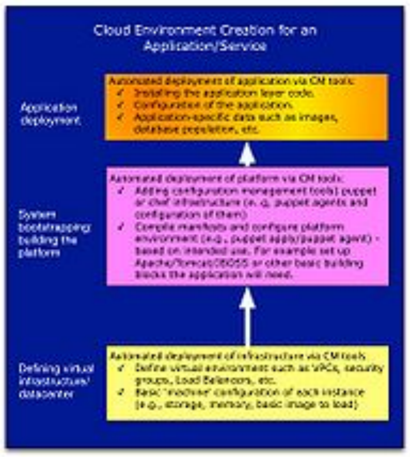
1. Facilities for migrating code from one environment to the next such as from test to development. This is most likely configuration of Git and Puppet.
2. Integration with other HUIT 'core services' (e.g., Identity and Access Management)
3. Autoscaling groups - not yet clear is this would be useful for individual engineers, though it will be required for test/verification.

Technologies for configuration and management (if any) are intended solely to stand up this Unix application development environment POC and neither the technologies nor the scripts that drive them should be thought of as other than drafts certain to either be significantly changed or discarded. This POC will implement and document the following in support of the identified use case.

1. CloudFormation Scripts drive the AWS environment. This also includes setting up a second availability zone used in the case of a failure of the first.
2. A stand alone VPC which may not have any connection to the Harvard

environment. It will not require nor use a VPN. Future reference implementations will likely add these facilities.
3. Security Groups will be created and all development systems will be in at least one.
4. Creation of at least one instance for each of the system types:
   1. A development system with all the tools defined in the use case.
   2. A single instance test system (For this POC, the difference between the development and test systems may be minimal).
   3. An initialized specialized server for build or other engineering project tasks.
5. CloudInit capabilities will be used to bootstrap Puppet.
6. Puppet will be used to configure the EC2 instances once created.
7. If time permits, we will create a Puppet Master in an autoscale group. Otherwise local Puppet agents on each instance will be 'feed' information required.
8. For the POC, the sequence of change management for configuration changes will be to apply changes in Git (stored using S3). After storage in Git a Post-Receive hook will be used. RPMs might be used in a reference implementation.
9. Facilities for spinning up and managing test environments. The idea is that engineers would be able to spin up entire test environments across availability zones and perhaps regions for some types of applications for testing purposes. It will be particularly helpful if the test environments are transitory which will help to ensure that set up for testing is externalized from and engineers local environment.
10. Basic CloudWatch monitoring/verification.

Deployment Activities/Phases in a Virtual Environment


Cloud Environment Creation for an Application/Service
Application deployment
Automated deployment of application via CM tools:
✓ Installing the application layer code.
✓ Configuration of the application.
✓ Application-specific data such as images, database population, etc.
System bootstrapping: building the platform
Automated deployment of platform via CM tools:
✓ Adding configuration management tools) puppet or chef infrastructure (e.g. puppet agents and configuration of them)
✓ Compile manifests and configure platform environment (e.g., puppet apply/puppet agent) - based on intended use. For example set up Apache/Tomcat/JBOSS or other basic building blocks the application will need.
Defining virtual infrastructure/ datacenter
Automated deployment of infrastructure via CM tools
✓ Define virtual environment such as VPCs, security groups, Load Balancers, etc.
✓ Basic 'machine' configuration of each instance (e.g., storage, memory, basic image to load)

Creating a Virtual

Environment

New architectures can also be used to re-examine operational approaches. The diagram in this section serves several purposes. First, it illustrates three logical groupings of activity to create a running application in an environment from creation of the virtual building blocks in a specific design pattern to installation and configuration of the running application. It also can be used to create a reasonable breakdown of tasks in the virtualized environment. Do not assume that there is specific/single organizational structure that could carry out these tasks. The organizational structure, while relevant, is not what drives this chart. In the context of the enterprise architecture effort, we are also using this to parcel out logical groups of tasks.

Retrieved from "https://wikis.fas.harvard.edu/huitarch/POCs_And_Reference_Implementations"