Design Decision

DECISION 1 : Use Codepipeline as CI/CD provider
In order to leverage AWS services and have better integration with the other services.

> **Upside**
> Out of the box integration with GitHub and Jenkins.
> Complex pipeline creation with custom actions is possible
>
> **Downside**
> Have minimal integration with other CI/CD providers.
> Custom action can be complex and takes time to understand.
> ===> **Workaround** Create multiple intermediate instance to handle non-default actions from CodePipeline
>
> **Alternative solutions**
> Use Jenkins or another CI/CD vendor to manage the whole pipeline.

DECISION 2 : Infrastructure Orchestrator
Delegate stack creation and minimal server configuration to CloudFormation.

> **Upside**
> All infrastructure specification in one single file. Allow resources referencing with a strong validation system.
> Handle infrastructure errors & inconsistencies.
> Handle stack update and dependencies. This is the basis for infrastructure as code with infrastructure versioning.
>
> **Downside**
> Instance configurations are harder to implement during launch.
> ===> **Workaround** Delegate instance configurations to a configuration management server like Chef or Opswork.
>
> **Alternative solution**
> Use SDK or API to generate resources

DECISION 3 : Store configuration settings externally
Store configuration files in GitHub or S3 and pull them based on context.

> **Upside**
> Avoid manual configuration
>
> **Downside**
> We have to store the whole context while only some values on specific files are changing.
> I my specific case, I had to store the files as *public* resources to avoid cross-account permission management. This is not a good practice
> ===> **Workaround** Only store files that contain the most changing configuration settings.
>
> **Alternative solutions**
> Use a templating system to update only the parts that are changing
> Create standard images and customize them with a configuration management solution (Packer + Chef)

DECISION 4 : Use Elastic Beanstalk as Deployment Provider
Even if we are using CloudFormation to manage the stack, it remains complex and error-prone. Elastic Beanstalk main purpose is to manage deployments.

> **Upside**
> Remove some complexity in the stack management.
> Provide an up-to-date environment with automated update management.
>
> **Downside**
> It might not be a great fit for complex environment with hundreds of instances and complex networking requirements.
> ===> **Workaround** Use eb configuration files to customize the environment for more complexity.
>
> **Alternative solution**
> Use Opsworks or CloudFormation

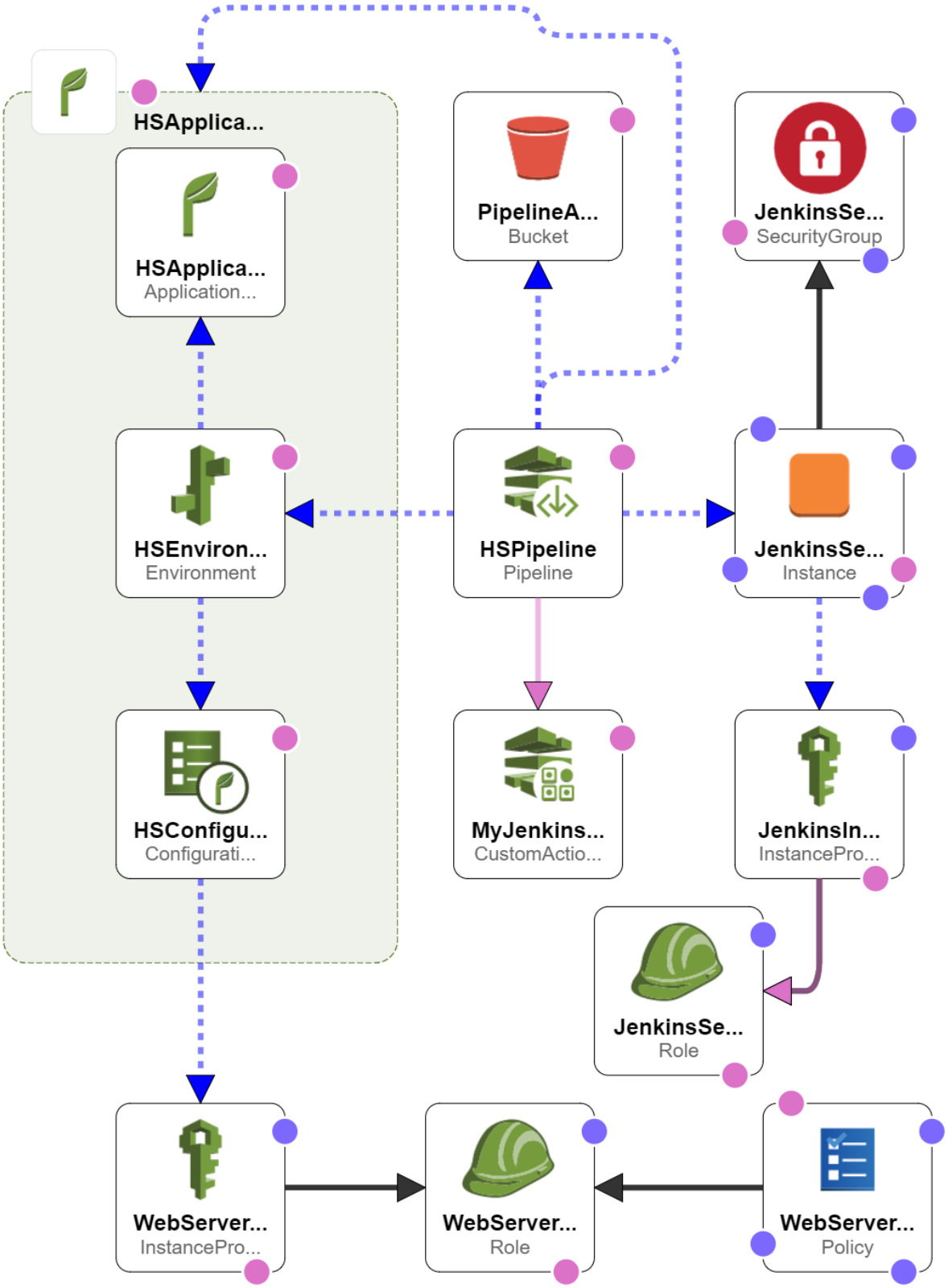
DECISION 5 : Self-Testing Server
Make the servers test themselves and notify the stack orchestrator (CloudFormation) when

something is wrong.

> **Upside**
> We stay in the logic of letting the orchestrator manage the infrastructure. CloudFormation will decide if it needs to rollback the stack, send a notification to the administrator or just move forward with the deployment.
>
> **Downside**
> If the instance doesn't reach the testing phase, we won't know why unless we log into the instance and check logs
> ===> **Workaround** Send logs to cloudwatch with monitoring agent.
>
> **Alternative solution**
> Use an external instance or local machine to perform the tests

---

Graphical Representation


HSApplica...
HSApplica...
Application...
HSEnviron...
Environment
HSConfigu...
Configurati...
PipelineA...
Bucket
HSPipeline
Pipeline
MyJenkins...
CustomActio...
JenkinsSe...
SecurityGroup
JenkinsSe...
Instance
JenkinsIn...
InstancePro...
JenkinsSe...
Role
WebServer...
InstancePro...
WebServer...
Role
WebServer...
Policy

Post-Mortem

1) Very interesting project

This was a very interesting project, mainly because of the multiple design possibilities. I could have spent days optimizing the process, trying to implement best practices. The number of tools and integration possibilities are impressive.

2) Easier with GUI & AWS Resources

Setting the pipeline with Cloudformation was straightforward. They have very good tools to create the template and we can create all the operations manually with a GUI called *CloudFormation Designer*.

Displaying the static page can be done directly with CodePipeline, source from GitHub and pushing to ElasticBeanstalk, which will handle the deployment. However, following my logic of delegating code build/testing to CI tools, I really wanted to integrate Jenkins in the pipeline.

3) Challenge : Jenkins instance standardization

This is where the challenge begun.

Since this is a central piece in the pipeline, creating a stable Jenkins instance with standard configuration for every deployment was a challenge. It was also a challenge to keep the Jenkins instance consistent for every deployment. Mainly because of the configuration settings and plugins.

My solution of storing the Jenkins jobs files in an S3 Bucket is a hack and is not a scalable solution. I fully understand the benefits of using an image creation framework like Packer and Chef to provision those instances. That way, we could provision multiple custom Jenkins instances to include in our pipeline.

4) Challenge : Test-Driven Infrastructure

The infrastructure testing was my biggest challenge. I found this great tool -Serverspec- which is doing exactly what I want. I then started playing a chicken & egg game, figuring out if I should create a new instance for testing purposes or go with the self testing.

I still think the self-testing logic is the best way to test the “logical” specification even if I wasn’t successful in fully automating it. With more time and stubbornness, I would have found a way to automate the whole testing / self-healing process I had in mind.

4) Time constraints vs. infinite optimization

I believe that time constraint is important. Looking for the perfect solution makes us lose time and delay delivery. That is a challenge for me as I am always trying to optimize. I really try to enforce time constraints to force myself to deliver, even if the solution is not perfect.

What to improve

- Create multiple Jenkins image with Packer and reference them when I create my instance instead of installing Jenkins from scratch every time. It would be faster and ensure better consistency and remove the need for storing files in S3 bucket.
- Use a template system for configuration files stored in S3. I we could populate the value dynamically when resources are created. This would make the code more modular. I know CloudFormation support *Mustache*.
- For more complex deployment, I would use CodeDeploy or Opswork instead of ElasticBeanstalk. EB is a great choice for non-complex code.
- I would tighten the security with better IAM roles and policy management. This is currently minimal.
- I would implement a better notification system when the stack is completed. Either using a Lambda function or SNS. This lambda function could then automate the testing process. This would remove the manual process.
- I would add more redundancy for my Jenkins instance. It is currently a single point of failure.

Conclusion

I learned a lot and it was fun. You did a great job with this project and I think you will be able to filter through all the required skills.

Good job!

> Written with StackEdit.