Virtual Design Master

Season 5 Challenge 3

Version 1.0

Chris Porter

Maybe we could automate this document…..?

Document History

Version	Date	Changes
1.0	19/07/2017	Version 0.3 bumped to version 1.0
0.3	18/07/2017	Everything else added.
0.2	15/07/2017	Conceptual Design added
0.1	14/07/2017	Document creation, addition of design brief

Glossary

Term	Definition
InfDevSecCleanCaterOps	The traditional devops team, but as we are short staffed they also cover Infrastructure and InfoSec, plus do the cleaning and catering.
Zombie Zone	The opposite of a Zombie Free Zone. i.e. where there are still zombies

Contents

1. Overview

1.1 Design Brief

The designs you have been working on through the first two challenges of this season have proven to solve many of the challenges faced by the Humanity Recovery Teams. We now have three sites running the HumanityLink software, and they have been secured. However, three sites will not be enough for long, as we begin to take more and more of the Earth back.

Some of the future sites slated for deployment have not yet been declared Zombie Free Zones, and are still considered dangerous. In addition, there are additional sites off-Earth being looked at for deployment. It is not always feasible to send a human team to do perform an infrastructure deployment for these reasons, and many others. We are also very short staffed, so the ability to perform centralized remote operations is also desired. Being able to deploy and manage the infrastructure and application in this manner is the next goal.

You must design and an environment to provision and manage your infrastructure. HINT: You may want to use open source products, but this is not a requirement or constraint. The design must include currently available products, it cannot utilize non-released features.

Choose and document how you will deploy and manage your infrastructure from the initial provisioning through to the continuous configuration management. Your submission must also include operations procedures for managing version control for your infrastructure and code. Every layer of your infrastructure from the cloud and virtualization layer up to the application must be included.

Practical walkthroughs and example code are valuable add-ons to your design document.

1.2 Design Summary

This design proposes a single tool to create, managed and run deployment pipelines for both the infrastructure and the HumanityLink application. That tool is Concourse, running in conjunction with GitLab, Terraform and Docker. Pipelines in Concourse continually trigger when new commits are

made to GitLab, and for the application they deploy to Pivotal Cloud Foundry. Our ultimate goal is to follow the ethos of a Pivotal haiku[i] that reads, from the perspective of a developer;

here is my source code
run it on the cloud for me
i do not care how

1.3 Design Scope

The design covers deploying and continually updating the HumanityLink infrastructure and the HumanityLink application. It does not cover the design of the infrastructure itself, or the design of the application.

2 Conceptual Design

2.1 Requirements

ID	Requirement
R001	It must be possible to perform infrastructure deployments without humans being on site
R002	The ability to perform centralized remote operations is also desired
R003	Version control for the infrastructure and application is required
R004	Initial provisioning through to continuous configuration management is required
R005	Operations procedures for managing version control for the infrastructure and code must be detailed
R006	Every layer of your infrastructure from the cloud and virtualization layer up to the application must be included
R007	Where possible, all applications and infrastructure services should be run on Cloud Foundry
R008	Infrastructure is potentially required to be deployed off earth

2.2 Constraints

ID	Constraints
C001	Only currently available products can be used, the utilisation of non-released products and features is prohibited.
C002	The application stack will run on Pivotal Cloud Foundry, due to its selection in the previous challenge
C003	AWS must be used, due to its selection in the 1st challenge

2.3 Assumptions

All assumptions should be validated to eliminate the risks associated with an incorrect assumption. Any non-validated assumptions should therefore also be documented as Risks with appropriate mitigations.

ID	Assumption
A001	As a critical infrastructure provider for the rebuilding of civilisation, AWS will support any strategy relating to further requirements of HumanityLink, such as deploying AWS Regions in space. AWS do not currently have a way to deploy regions to space however and so one must be designed for them.
A002	Whilst it has been suggested we may want to use open source products, the usage of close source products is still a viable design option
A003	The latency and bandwidth means that Zombie Zone and off earth deployments would be standalone sites, which would not participate in the replication and failover of the two main sites

2.4 Risks

ID	Risk	Mitigation
K001	Some potential sites for future expansion have not yet been declared Zombie Free Zones	Do not send human teams to deploy new infrastructure. Infrastructure must be able to be deployed without needing humans on site.
K002	Assumption A001 is incorrect and AWS will not support deploying additional regions	For basic infrastructure we would use VxRail to support deployments of PCF in non AWS locations. For IoT we would need to use an alternate solution based, ideally based on PCF
K003	Assumption A002 is incorrect and we only have access to open source applications, either from the start or at some point in the future	Use open source products whee possible. We can continue to use Cloud Foundry as it is open source. Whilst AWS is based on the open source hypervisor Xen, the management layer is unknown and so could be considered as closed source. If this meant the usage of AWS were unavailable, and VMware as well, we would use OpenStack, as it has good support for Cloud Foundry.
K004	A circular dependency could occur If infrastructure services required to maintain PCF are also running on PCF, as stated in Requirement R007	Any infrastructure components that are used to support PCF must run independantly of PCF.

K005	Assumption A003 is incorrect, and in particular Zombie Zone sites need to particiapte in replication and failover	Bribe application and infrastructure teams with donuts to ensure the assumption associated with this risk is correct, as the architect realised this design issue too close to submission to come up with any other mitigation.

ii

3 Logical Design – Zombie Zone and Off Earth Deployments

Design Decision ID	LD001
Description	AWS regions with three availability zones would deployment to other planets or zombie zones as required.
Justification	AWS is the core infrastructure for HumanityLink and a 3 site architecture is required.
Conceptual Design Reference	R008, C003, A001

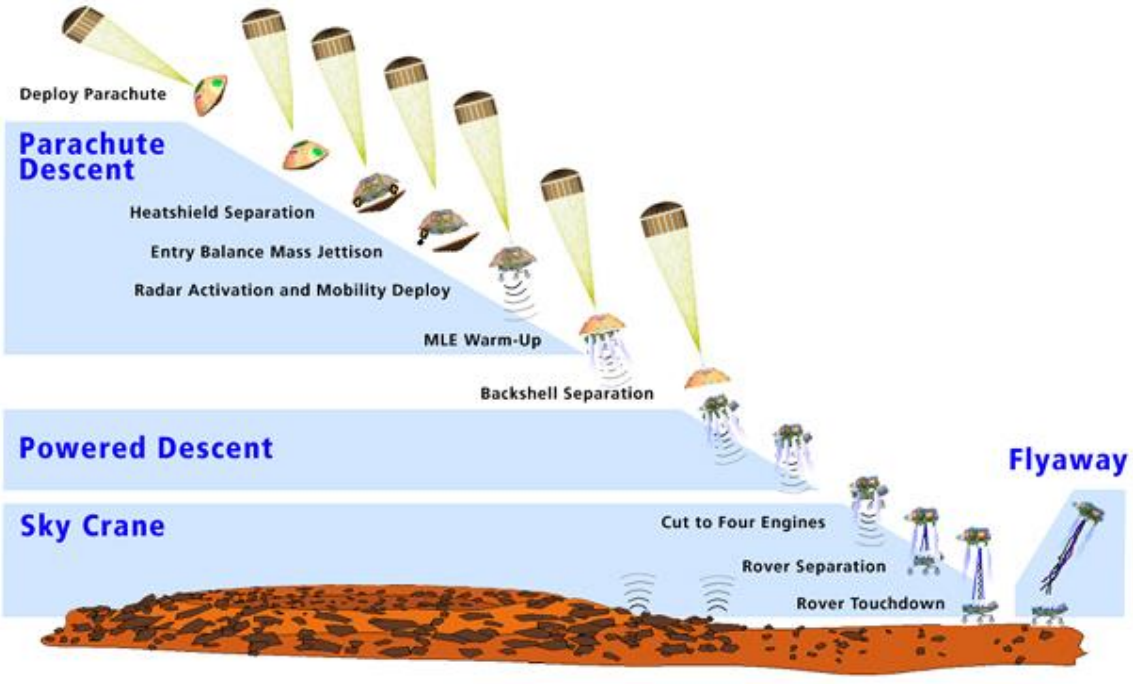


Figure 1; The stages the data centre container would go through as it was delivered to the planet's surface.[iii]

Design Decision ID	LD002
Description	To allow for datacentres to be deployed in areas of the Earth where it is unsafe for humans or off earth on planets such as Mars or the moon, pods of datacentre hardware will be deployed by rocket. To ensure the impact of landing is within acceptable boundaries, a parachute and reverse thruster based delivery mechanism will be used.
Justification	The rocket, parachute, reverse thrust deployment worked successfully for the Mars Rover, which weighed 900Kgs[iv], so we should be able to use this mechanism to deploy multiple servers and switching equipment in an small container (based on VxRail E Series weiging 18.5Kg each[v]). They would be powered by solar and deployed on to areas of planets where cooling

	requirements can be minimised. This same mechanism would be used to deploy on Earth to areas where humans could not go.
Conceptual Design Reference	R001, R008, C001, A003, K001

Figure 2; This giant robot fixes undersea broadband cables[vi]

Design Decision ID	LD003
Description	Cable laying robots would be used to lay fibre between DC containers delivered off site or in zombie zones
Justification	Robots are already used to lay cables in sewers and to repair undersea cables, so in this configuration they would be deployed in each container and then upon landing, the robot would trundle to the next container to terminate and light up high speed links between the containers
Conceptual Design Reference	R001, R008

4 Logical Design; Infrastructure

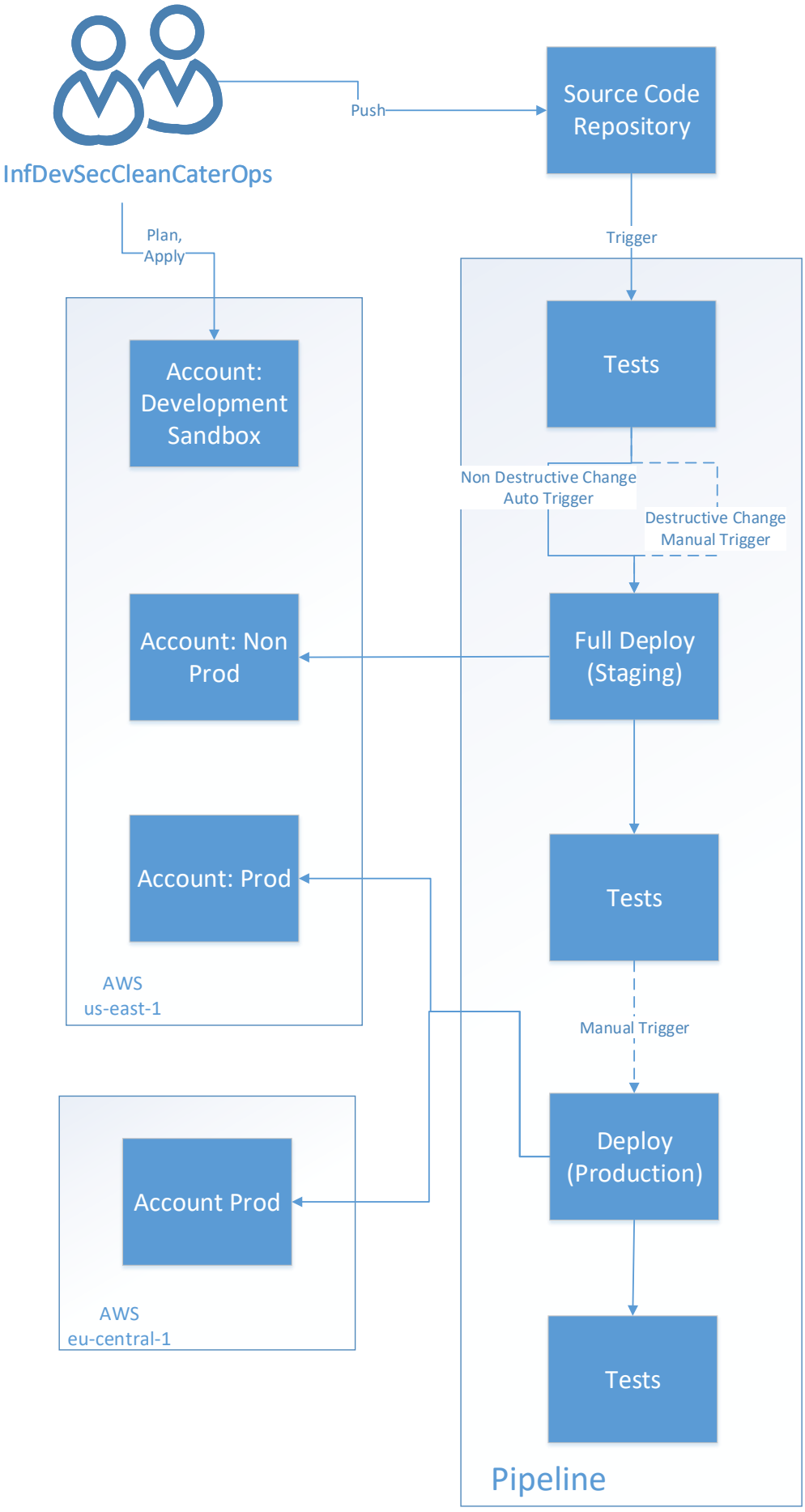
The infrastructure in AWS will be deployed from code, using a deployment pipeline. The code would be stored in a version control system to allow for the deployment to be automated. Cloud Foundry infrastructure can be built using code on AWS[vii]. If we are still able to use closed source software, Pivotal Ops Manager would be deployed to manage the deployment and updating of our Pivotal Cloud Foundry installations.

Design Decision ID	LD004
Description	All Infrastructure must be defined in code and stored in version control
Justification	Defining infrastructure in code allows for it to be version controlled and for infrastructure deployments to be automated.
Conceptual Design Reference	R003, R002

Design Decision ID	LD005
Description	A deployment pipeline will be used to deploy the HumanityLink infrastructure from the infrastructure code
Justification	Using a deployment pipeline ensures a consistent repeatable deployment process every time a change is made to the infrastructure. It allows for infrastructure deployments to be performed in a completely automated fashion and managed from a centralised location, from initial deployment through to every update. It allows us to properly use the term 'Infrastructure as code'
Conceptual Design Reference	R001, R002, R004

The deployment pipeline for infrastructure combines automated triggers with testing to ensure an efficient but safe deployment. It would work as follows (also shown in diagram after numbered list);

1. A trigger fires when a new version of the infrastructure code is committed to the source code repository.
2. Basic test(s) are run to ensure the code doesn't errors before it deploys (i.e. terraform plan). This test will also determine whether the new code will cause destructive changes or not.
3. If destructive changes are detected, a notification is sent and the deployment step must be triggered manually. Otherwise, the next step is triggered automatically
4. The code is deployed to the staging environment within the non prod AWS account
5. Tests are run against the infrastructure to ensure it is still functioning as expected
6. If the tests are successful, then a notification is sent
7. A manual trigger deploys the code into Production in both AWS regions
8. The pipeline ends with tests to ensure the production deployment was successful.


InfDevSecCleanCaterOps
Push
Source Code Repository
Trigger
Plan, Apply
Tests
Account: Development Sandbox
Non Destructive Change Auto Trigger
Destructive Change Manual Trigger
Account: Non Prod
Full Deploy (Staging)
Account: Prod
Tests
AWS us-east-1
Manual Trigger
Account Prod
Deploy (Production)
AWS eu-central-1
Tests
Pipeline

Design Decision ID	LD006
Description	For any services deployed outside of PCF, containers will be used
Justification	Containers allow for rapid and consistent deployment of services
Conceptual Design Reference	K004

Design Decision ID	LD007
Description	A Git based source code repository server will be deployed
Justification	Git allows for global distributed source code repositories and is the de facto standard for software source control[viiiix], which ensures the widest possible selection of other infrastructure services as they most likely support Git. It also ensures the best chance of familiarity for operators and developers, reducing time to start working on HumanityLink and its infrastructure. Plus if the world ever sorts itself out, Git is good to have on your CV ;)
Conceptual Design Reference	R003

Design Decision ID	LD008
Description	The Git based source code repository server will be deployed, but running in container(s) rather than on PCF
Justification	Avoids circular dependencies if deployment tool for PCF uses source code in the source code server.
Conceptual Design Reference	K004

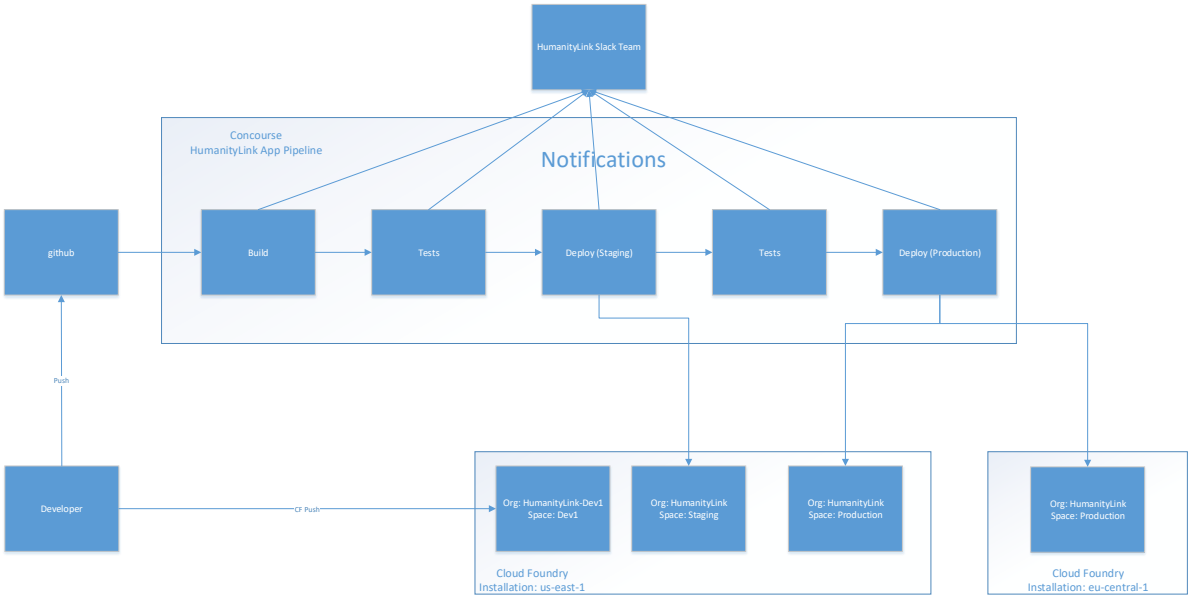
Design Decision ID	LD009
Description	A pipeline will be run on a schedule to check the current infrastructure against the infrastructure defined in code. This will notify based on any discrepancies.
Justification	Ensures configuration drift is minimised
Conceptual Design Reference	R005

5 Logical Design; Applications

To meet the conceptual design, a deployment pipeline will be implemented for CI/CD of the HumanityLink application.

Design Decision ID	LD010
Description	Deployment pipeline will used to deploy HumanityLink application
Justification	Using a deployment pipeline ensures a consistent repeatable deployment process every time a change is made to the application. It allows for application deployments to be performed in a completely automated fashion and managed from a centralised location, from initial deployment through to every update.
Conceptual Design Reference	R001, R002, R003, R004, R006

Design Decision ID	LD011
Description	The deployment pipeline tool will not be deployed on PCF and instead will be deployed in containers
Justification	Avoids circular dependencies if deployment tool is also used to deploy PCF
Conceptual Design Reference	K004



Concourse pipeline with notifications to Slack;

1. Git commit based trigger
2. Build

3. Test
4. Deploy to staging
5. Test
6. Deploy to Prod in both locations

6 Logical Design; IoT and Lambda

Design Decision ID	LD012
Description	Lambda functions to control IoT devices will be deployed using a deployment pipeline
Justification	Managing code for IoT devices is critical to ensure they carry out terraforming activities successfully. Lambda functions should be versioned and deployed in the same way as other application code.
Conceptual Design Reference	R001, R002, R003, R004, R006

7 Physical Design; Infrastructure

The infrastructure in AWS will be deployed using Terraform, using a pipeline in Concourse. Concourse has been chosen as it has strong support for Cloud Foundry but also supports Terraform, so the same tool can be used for both application and infrastructure components.

The initial Pivotal Cloud Foundry infrastructure would be built using Terraform[x] via a Concourse deployment pipeline. This spins up a Pivotal Ops Manager EC2 instance which can then be used to manage the deployment and updating of Pivotal Cloud Foundry[xi].

Design Decision ID	PD001
Description	Terraform will be used to define our infrastructure in code
Justification	Terraform supports running a test before applying the code, supporting our logical design for the infrastructure deployment pipeline. It is also open source and supports multi infrastructure providers.
Conceptual Design Reference	R001, R002, R003, R004, R006, K004

Design Decision ID	PD002
Description	Concourse will be used to configure and run both infrastructure and application deployment pipelines
Justification	Concourse is open source and has strong integration with Pivotal Cloud Foundry. It also supports terraform as a resource so it can deploy the infrastructure[xii].
Conceptual Design Reference	R001, R002, R003, R004, R006, K004

The following Docker compose file would create the 3 containers required to run Concourse. This has been taken from http://concourse.ci/docker-repository.html, and edited to make the data for the database container persistent, as per https://gist.github.com/SeerUK/59fc5ae5371cab37aafa46cff54c14bf

```
version: '3'


services:
  concourse-db:
    image: postgres:9.5
    environment:
      POSTGRES_DB: concourse
      POSTGRES_USER: concourse
      POSTGRES_PASSWORD: changeme
      PGDATA: /database
        volumes:
```

```
          concourse-db-data:/database


  concourse-web:
    image: concourse/concourse
    links: [concourse-db]
    command: web
    depends_on: [concourse-db]
    ports: ["8080:8080"]
    volumes: ["./keys/web:/concourse-keys"]
    restart: unless-stopped # required so that it retries until conocurse-db comes up
    environment:
      CONCOURSE_BASIC_AUTH_USERNAME: concourse
      CONCOURSE_BASIC_AUTH_PASSWORD: changeme
      CONCOURSE_EXTERNAL_URL: "${CONCOURSE_EXTERNAL_URL}"
      CONCOURSE_POSTGRES_HOST: concourse-db
      CONCOURSE_POSTGRES_USER: concourse
      CONCOURSE_POSTGRES_PASSWORD: changeme
      CONCOURSE_POSTGRES_DATABASE: concourse


  concourse-worker:
    image: concourse/concourse
    privileged: true
    links: [concourse-web]
    depends_on: [concourse-web]
    command: worker
    volumes: ["./keys/worker:/concourse-keys"]
    environment:
      CONCOURSE_TSA_HOST: concourse-web


volumes:
  concourse-db-data:
    driver: local
```

Design Decision ID	PD003
Description	GitLab will be used as the server for infrastructure and application source code repositories
Justification	GitLab is open source and can be deployed locally without relying a SaaS solution
Conceptual Design Reference	R001, R002, R003, R004, R006, K003

GitLab can be deployed from the container in Docker Hub found at https://hub.docker.com/r/gitlab/gitlab-ce/

Design Decision ID	PD004
Description	Docker will be used for containers
Justification	Like using Git for source code, Docker is the de facto standard for containers which ensures the widest possible selection of pre built container images and operator familiarity. Concourse and GitLab are also supplied as Docker images. Again, like Git, if the world ever sorts itself out, Docker + Git is good to have on your CV. In fact, we could probably get a job with the bad actor as all the cool/sensible malware writers are using devops tooling these days.
Conceptual Design Reference	K003, K004

Design Decision ID	PD005
Description	AWS ECS will be used to host Docker containers
Justification	Using AWS ECS makes it easy to run docker containers in a robust, scalable and reliable manner without having to deploy the underlying infrastructure. If AWS were unavailable, then a container host would be built. Due to the limited number of containers, a container orchestration tool would not be required.
Conceptual Design Reference	K004

Design Decision ID	PD006
Description	Terraform will be used to deploy the instances that run MongoDB, and the terraform deployment will initiate a script to install MongoDB[xiii]. A separate Terraform pipeline will trigger a script[xiv] to fetch and install MongoDB updates.
Justification	Terraform is being used as the standard for deploying infrastructure.
Conceptual Design Reference	R001, R002, R003, R004, R006, K003

8 Physical Design; Applications

The following YAML creates a basic pipeline for deploying the HumanityLink app. It is non functioning as the build and deployment steps have been added only as placeholders.

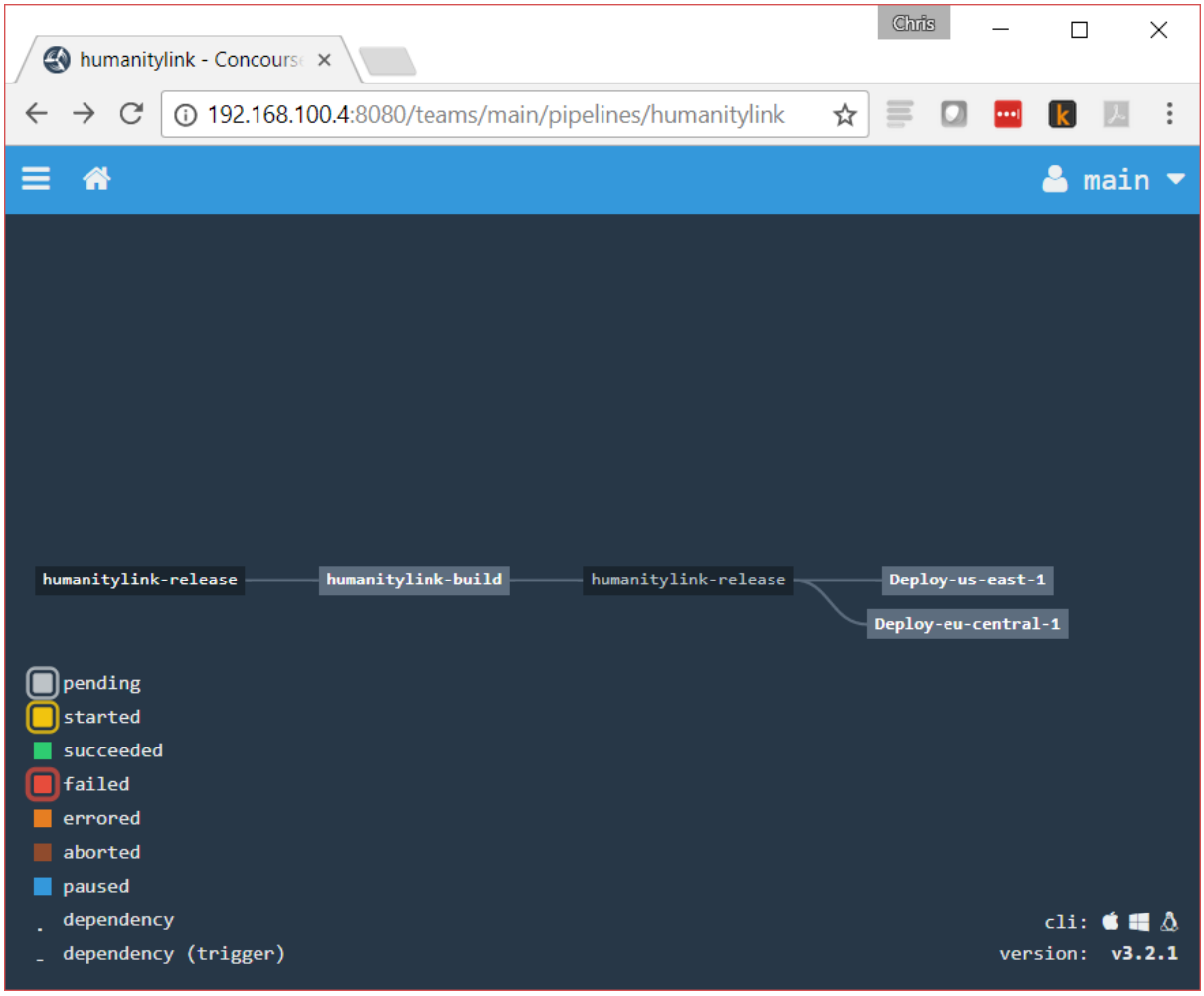
```
resources:
- name: humanitylink-release
  type: git
  source:
    uri: https://gitlab.humanitylink.com/HumanityLink/hl.git

jobs:
- name: humanitylink-build
  plan:
  - get: humanitylink-release
    trigger: true
  - task: humanitylink-build
    file: placeholder-task.yml

- name: Deploy-us-east-1
  plan:
  - get: humanitylink-release
    trigger: true
    passed:
      - humanitylink-build
  - task: Deploy-us-east-1
    file: placeholder-task.yml

- name: Deploy-eu-central-1
  plan:
  - get: humanitylink-release
    trigger: true
    passed:
      - humanitylink-build
  - task: Deploy-eu-central-1
    file: placeholder-task.yml
```

This creates the following concourse pipeline;



The YAML for this pipeline originated from Pivotal's Concourse Pipeline Samples[xv]

9 Physical Design; IoT and Lambda

Design Decision ID	PD007
Description	Terraform will be used to deploy Lambda. As part of a Concourse pipeline, the lambda function would be zipped and then terraform would deploy the zip file[xvi].
Justification	Terraform is being used as the standard for deploying infrastructure.
Conceptual Design Reference	R001, R002, R003, R004, R006, K003

10 Implementation Guide

The following steps would be followed to deploy the infrastructure;

1. Use Terraform standalone to deploy the base AWS infrastructure for one region (see Challenge 1 for base terraform code)
2. Deploy GitLab in a container
3. Commit terraform code and state file to GitLab
4. Deploy Concourse containers
5. Deploy terraform concourse resource container
6. Create a pipeline for infrastructure deployments. Run the pipeline against the terraform code committed in step 3 to check the automated process.
7. Create and run pipeline for PCF initial infrastructure setup and Pivotal Ops Manager deployment
8. Use Pivotal Ops Manager to deploy Pivotal Cloud Foundry
9. Create HumanityLink application deployment pipeline for one region in Concourse
10. Commit HumanityLink source code to GitLab to trigger first deployment to PCF
11. Use Infrastructure Deployment Pipeline to deploy second region
12. Use pipeline to deploy PCF to second region
13. Update HumanityLink pipeline to point to second region

11 Appendix A; References

[i] Pivotal Cloud Foundry's Roadmap For 2016
https://content.pivotal.io/blog/pivotal-cloud-foundry-s-roadmap-for-2016

[ii] Installing Pivotal Cloud Foundry on OpenStack
https://docs.pivotal.io/pivotalcf/1-11/customizing/openstack.html

[iii] Entry events from parachute deployment through powered descent ending at sky crane flyaway
https://en.wikipedia.org/wiki/Mars_Science_Laboratory#/media/File:20090428MSLEntry2.jpg

[iv] Mars rover weight
https://mars.nasa.gov/msl/mission/spacecraft/

[v] VxRail Weight;
https://www.emc.com/collateral/specification-sheet/vxrail-4.0-spec-sheet.pdf

[vi] This giant robot fixes undersea broadband cables
http://www.wired.co.uk/article/fixing-submarine-cables

[vii] Preparing to Deploy on Amazon Web Services
https://docs.cloudfoundry.org/deploying/aws/

[viii] *"By far, the most widely used modern version control system in the world today is Git."*
https://www.atlassian.com/git/tutorials/what-is-git

[ix] *"Git is the most commonly used version control system today and is quickly becoming the standard for version control."*
https://www.visualstudio.com/learn/what-is-git/

[x] Terraforming PCF
https://github.com/pivotal-cf/terraforming-aws

[xi] Using Terraform, Concourse, and om to Continuously Deploy Pivotal Cloud Foundry's Elastic Runtime
https://engineering.pivotal.io/post/release-engineering-deploying-ert/

[xii] Concourse Terraform resource
https://github.com/ljfranklin/terraform-resource

[xiii] Install MongoDB Enterprise on Amazon Linux
https://docs.mongodb.com/manual/tutorial/install-mongodb-enterprise-on-amazon/

[xiv] Terraform Provisioners
https://www.terraform.io/docs/provisioners/index.html

[xv] concourse-pipeline-samples/concourse-pipeline-patterns/parameterized-pipeline-tasks/package-tutorials.yml
https://github.com/pivotalservices/concourse-pipeline-samples/blob/master/concourse-pipeline-patterns/parameterized-pipeline-tasks/package-tutorials.yml

[xvi] How to deploy an AWS Lambda with Terraform
https://seanmcgary.com/posts/how-to-deploy-an-aws-lambda-with-terraform