2017

Challenge 3

APPLICATION EXPANSION

KYLE JENNER

TERRAFORMING SPACE AGENCY

1 Contents

2 Contact Information & Document Control

The primary contacts for questions and discussions regarding this proposal are:

2.1 Document Information

Title	**Application Expansion**
Version	1.1
Author	Jenner, Kyle
Distribution Date	18/07/2017
Number of Pages (Excluding Cover)	25

3 Executive Summary

3.1 Document Purpose

The purpose of this document is to outline the key elements and design decisions which make up the proposed infrastructure design.

3.2 Project Overview

Terraforming Space Agency (TSA) recently designed and deployed the new infrastructure to support HumanLink version 2.0 application and to secure the environment. The humanity recovery teams are looking to expand further than the original 3 site architected.

Sites on earth have been highlighted as possible locations to expand the HumanLink application but none of them have been declared zombie free zones presenting extreme challenges. TSA have been assigned the task to not only expand the infrastructure to another site but to also do it without any human visits to the locations.

Centralized management to the additional site is critical. This proposed deployment and management model will be suggested for further sites including sites outside of Earth. Where possible every layer of the infrastructure must be included.

3.3 Design Qualities

The following design qualities will be referenced.

Qualities	Ref	Example
Availability	DQA	System up time to deliver SLAs.
Manageability	DQM	Simplified management layer to reduce overall efficiency.
Performance	DQP	Ensure system performance to meet project requirements.
Recoverability	DQR	Ability to recover from a failure.
Security	DQS	Authorization and access to the system.

4 Conceptual Design

4.1 Requirements

Requirement	Ref
Identify a site location for expansion.	REQ001
Document a human free automated deployment method.	REQ002
Additional site must be centrally managed.	REQ003
The solution must provide and method for continuous configuration management.	REQ004
The solution must provide operational procedures for version control.	REQ005
Include practical walkthroughs and example code.	REQ006
The solution must consider an expansion beyond Earth.	REQ007

4.2 Constraints

Constraint	Ref
The solution can only use products that are generally available.	CON001
Identified sites cannot be physically visited by human staff.	CON002

4.3 Risks

Risk	Impact	Mitigation	Ref
Zombies are still out there!	High	Robots must be used to initiate first contact with the site and have Rick and Carl Grimes on standby.	RIS001

4.4 Assumptions

Assumption	Ref	Additional Information
VMware on AWS no longer is available for new sites.	ASU001	Products that are generally available can only be used (CON001).
AWS locations and facilities still exist and can be used.	ASU002	AWS datacentres still exist but are potentially over run by zombies, the facilities can still be used if we can get to them.
Some of the robot army can be used.	ASU003	Some of the existing robot army can be redeveloped and used to physically visit the locations.
The 3 existing sites are functioning as designed.	ASU004	The existing sites are still working as normal.
Zombies actually do love haggis.	ASU005	Contrary to popular belief zombies do actually love haggis….what's not to love!

4.5 Conceptual Design Overview

The following conceptual design will be a very high level view on what the solution will look like after completion. The conceptual design does not include any sizing figures, vendors or product specifications.

The new datacentre location is critical to the human terraforming mission, the application needs expanding and to another site on Earth.

It is suspected the datacentres were overrun by zombies given the worlds previous encounters and what's worse it is rumoured some of the surviving humans were saved from the zombie infection by altering genetics and creating cyber-enhanced cyborgs. Rumours are these cyborgs live is small communities with rouge humans around the world, not even Daryl and Carol can stop them.

Given this, it is extremely unsafe to send any human party to abandoned datacentres. Instead some of the robot army have been re-programmed to go to these sites, infiltrate them and load up the code required to get the HumanLink application up and running. Once inside the robots will prevent any zombie, cyborg or human from entering to the best of their ability.

The robots will be programmed to establish links back out to the internet and allow the centralised sites to be able to deploy and manage the new sites. Each robot will be armed with bags of haggis as a last-ditch defence mechanism should they find themselves in zombie trouble (AUS005).

It's a brave task for these little robots and humanity salutes you.

5 Logical Design

The following logical design will take the conceptual design and put together a solution that will use technologies to meet the requirements. Each section will list a design decision and link it to the requirement for reference. Each design decision has considered the risks and constraints highlighted above.

5.1 Logical Design Overview

The new datacentre location is critical to the human terraforming mission, existing infrastructure runs on AWS either natively or with VMware on AWS. The latter is not available to this project (CON001) and none of the other AWS regions and AZs are online.

An existing abounded AWS site has been selected to be restored at this stage, should the method be successful further missions to further sites on Earth or beyond will be possible.

The robots will infiltrate the datacentre and connect the internet links back up, once restored deployment of the application and its required infrastructure needs to be automated as the mission is too delegate for people to connect and run manual tasks, the method needs to be fully verified on the working sites before moving out to other high risk sites.

Following initially deployment the application must be maintained and managed centrally as well as following standard change control measures. Continuous Integration (CI) and Continuous Delivery (CD) practices will be introduced as the standard for development testing and application delivery.

Docker will be integrated into the CI pipeline. The immutability of Docker images ensures a repeatable deployment with what is developed and tested through CI to what is run in production.

At this stage the design is focussed for an additional site and infrastructure to deploy the HumanLink application further. The VMware on AWS environment used by the staff will not be expanded. Remote scientists, engineers and developers will continue to work as normal.

Changes to the application and development method will be required to be able to make it truly portable when looking to move to outside of Earth, HumanLink application cannot rely on one provider alone given the unknowns around the threats to the datacentres.

No longer can we rely on DynamoDB for the scalability just in case in the next abandoned datacentre is not an AWS datacentre. An open source NoSQL database will be used instead and added into the application stack.

5.2 Infrastructure Logical Design

Given their size and infrastructure a AWS datacentre will be selected for the site expansion. Robots will be deployed to the selected region and infiltrate the datacentre. Once back online the main sites will be able to communicate with them.

The required infrastructure will be deployed as code from a tried and tested deployment method.

DDN001	Decision – Reactive abandoned AWS datacentre.
	Justification – To make use of existing infrastructure without looking to deploy from scratch.
	Type – DQA, DQM, DQP
	Impact – Once datacentre is back online infrastructure needs to be configured for use.
	Associated Risk – Infrastructure and application deployment will be AWS specific and not portable if the next abandoned site is not AWS.
	Risk Mitigation – Use deployment and application methods that do not rely on the underlying infrastructure.
	Reference – REQ001 ASU002, AUS03, ASU04

Terraform will be leveraged to deploy the AWS infrastructure ready for the new site application, deploying infrastructure as code will allow for orchestrated deployment of the new AWS infrastructure.

Terraform will be used to plug into the VMware infrastructure the developer use to be able to test out deployments. Terraform deployment codes can be ported to any infrastructure should it be required.

An orchestration tool was picked over configuration management tool on the basis the application is running on Docker images where the software has already been installed and configured. Every change will be a new deployment of the new image then a removal of the old one. This reduces the possibility of configuration drift.

Terraform is a declarative approach where the end state is declared and Terraform will figure out how to get to the end state, for example deploying 10 instances initially then changing the script to 20 will only add another 10 instances and not an additional 20.

Additionally, Terraform uses a client-only architecture where no additional servers or authentication other than the native provider details are required.

Teams can work together to remotely manage changes to the environment by sharing state files on remote storage or a Git repository. When a script deploys, it will right the results of the deployment and a state of the deployment to a state file, when another script is run this state file is checked and compared to know if anything new will be deployed.

Using a Git repository allows teams to collaborate and track changes, it can also be integrated further into the CI/CD process.

Remote config will not lock the state file however, the teams will need to work together and plan accordingly to make sure no two changes are made at once. Terraform plan command can be run to know exactly what will be changed before deployment.

DDN002	Decision – Use Terraform to deploy new infrastructure.
	Justification – To automate the deployment of the infrastructure.
	Type – DQM
	Impact – Deployment will be created in code; no web management page will be used.
	Associated Risk – Changes to the code could affect production infrastructure.
	Risk Mitigation – Terraform offers the concept of an execution plan showing what will be changed without actually running the plan, this can be reviewed before deployment. Code will be stored centrally on GitHub for version control and management between the team.
	Reference – REQ002, REQ003, REQ004, REQ005 ASU04

5.3 Application Logical Design

The original application running in native AWS required DynamoDB but moving forward any further expansion may not be able to run on AWS services. The application will be migrated to an open source NoSQL database using Cassandra.

Following the decision to change the application database TAS researched various case studies published before the zombie outbreak around performance and scalability. University of Toronto compared performance on different NoSQL platform where Cassandra came out on top. Neflix published a paper on the scalability around Cassandra and we all miss Netflix in this world.

DDN003	Decision – HumanLink application to use Cassandra.
	Justification – Can store huge datasets ideal for collection data from sensor arrays at each terraforming site while providing scale and reliable cross-datacentre replication.
	Type – DQA, DQP
	Impact – The application needs to be redeveloped to support Cassandra and data migrated from DynamoDB.
	Associated Risk – Losing a node could result in data loss.
	Risk Mitigation – Cassandra will be deployed in a cluster and the replication factor configured to ensure data is stored on multiple hosts.
	Reference – REQ007

HumanLink application will be developed to run as a containerised application using Docker images, these Docker images will be used and run in Amazon EC2 Container Services (ECS) as stipulated in the original infrastructure design. Due to expansion of the application CI/CD measures will be introduced.

CI/CD merges development with testing allowing the developers at each site to work collaboratively which can be submitted to centralised development platform such as GitHub. This allows the code to be tested as early as possible to catch bugs early in the development lifecycle.

By integrating Docker CI process ensures repeatable deployment from testing to production using immutable images. Developers will submit code to GitHub and test the code before going into deployment.

Jenkins will be added to streamline this process, Jenkins can run automatically pull the code once it had been uploaded then Jenkins can deploy the code and test. If the tests are successful, the image is then pushed up to Amazon Container Registry (ECR).

Jenkins can be extended further by adding Terraform onto the server allowing for a Jenkins pipeline that can pull down changes from GitHub and test infrastructure deployments. The success or failure of the test can then notify the team.

DDN004	Decision – Introduce Jenkins to the development lifecycle.
	Justification – To streamline application testing and deployment.
	Type – DQM
	Impact – Jenkins is required to be deployed and configured with GitHub.
	Associated Risk – Reliance on GitHub in this uncertain world.
	Risk Mitigation – Use another Git repository or deploy GitHub on-premises.
	Reference – REQ004, REQ005 ASU004

DDN005	Decision – Configure Jenkins to push to Amazon ECR.
	Justification – To streamline application testing and deployment.
	Type – DQM
	Impact – Amazon ECR must be configured and the plugin must be integrated with Jenkins. Amazon ECR supports Docker registry API allowing the use of Docker CLI commands.
	Associated Risk – Future datacentres may not be running AWS.
	Risk Mitigation – Private or hosted Docker registries can be used with Jenkins.
	Reference – REQ004, REQ005 ASU004

6 Physical Design

The following will outline the physical design using the conceptual and logical design decisions above.

6.1 HumanLink Application

The following section will detail the application deployment details for the HumanLink application, see section 7.1 for the deployment script. The deployment script will change depending on the environment, the below covers the abandoned AWS site recovered in Ireland - **eu-west-1**.

6.1.1 Infrastructure

AWS details

Region	AZ
eu-west-1	eu-west-1a
	eu-west-1b
	eu-west-1c

VPC details

Name	IPv4 CIDR	IPv6 CIDR	Tenancy
vdmVPC	200.0.0.0/16	-	Default

Networking details

Attribute	Specification
Role	Subnet
Name	ecsvdmPubSN0-0
VPC	vdmVPC
IPv4 CIDR	200.0.0./24
IPv6 CIDR	-
Available IPv4	247
AZ	eu-west-1a

Attribute	Specification
Role	Routing Table
Name	vdmPubSN0-0RT

VPC	vdmVPC
Destination	0.0.0.0/0
Target	ecsvdmIG

Security details

Attribute	**Specification**
Role	Security Group
Name	vdm_load_balancers
Description	Allows all Traffic
Ingress	From port = 0 To port = 65535 Protocol = tcp CIDR block = 0.0.0.0/0
Egress	From port = 0 To port = 65535 Protocol = tcp CIDR block = 0.0.0.0/0
Role	Security Group
Name	humanlink_ecs
Description	Allows all Traffic
Ingress	From port = 0 To port = 65535 Protocol = tcp CIDR block = 0.0.0.0/0
Egress	From port = 0 To port = 65535 Protocol = tcp CIDR block = 0.0.0.0/0

Amazon EC2 Container Service

Name	**Services**	**Tasks**	**Container Instances**
vdm-ecs	1	4	4

Attribute	Specification
Role	ECS Cluster
Name	vdm-ecs
Auto scaling Group	vdm-ecs-as Min size = 2 Max size = 5 Desired = 4 Health Check = EC2
Launch configuration	esc-lc Instance type = t2.micro Security Group = humanlink_ecs IAM Instance Profile = ecs_ip
aws_iam_role	ecs_instance_role = ec2.amazonaws.com ecs.amazonaws.com ecs_scheduler_role = ec2:Describe elasticloadbalancing:DeregisterInstancesFromLoadBalancer elasticloadbalancing:Describe elasticloadbalancing:RegisterInstancesWithLoadBalancer

Terraform will be deployed using remote configs running on GitHub. Different parts of the infrastructure will reside in different folders to provide some sort of isolation and management. The folder layout for the infrastructure will be as follows

- Production
  - VPC
  - Services
  - Roles
- Staging
  - VPC
  - Services
  - Roles

6.1.2 Application

Following the infrastructure deployment, the script follows on with the deployment of the application. The application is a basic webpage that says "hello-world" and is published externally once the cluster node start up. This is achieved by running tasks. The application is pulled from Docker Hub and is called **training/webapp:latest**.

This is to demonstrate the deployment.

Attribute	Specification
Role	Task Definition
Name	helloworldcontainer

Container Definition	CPU = 128
	Container Port = 5000
	Host Port = 80
	Memory = 128
	Image = training/webapp:latest
	Name = helloworld

Attribute	Specification
Role	ECS Service
Name	helloworld
Cluster	vdm-main
Desired Count	4
Task Definition	helloworldcontainer

Additional Terraform folders will be created to manage the application and changes

- Production
    - ECS
    - Tasks
    - Roles
- Staging
    - ECS
    - Tasks
    - Roles

6.2 Cassandra

The following details the deployment of the Cassandra database. This is the details of the initial deployment in AWS, future deployments may not be in AWS.

# Instances	# AZ	Replication Factor	Snitch
3	3	3	EC2MultiregionSnitch

EC2 details

Attribute	Specification
Role	Cassandra Server
Name	db-cas-01
Instance Details	Instance Type = m4.2xlarge vCPU = 8 Memory = 32GB Network Performance = High Instance Storage = EBS only
VPC	db_VPC
Subnet	Name -AZ1 CIDR – 10.0.0.0/18 Auto-assign IP = Yes
Network Interface	Eth0

Attribute	Specification
Role	Cassandra Server
Name	db-cas-02
Instance Details	Instance Type = m4.2xlarge vCPU = 8 Memory = 32GB Network Performance = High Instance Storage = EBS only
VPC	db_VPC
Subnet	Name -AZ2 CIDR – 10.0.64.0/18 Auto-assign IP = Yes

Network Interface	Eth0

Attribute	Specification
Role	Cassandra Server
Name	db-cas-03
Instance Details	Instance Type = m4.2xlarge vCPU = 8 Memory = 32GB Network Performance = High Instance Storage = EBS only
VPC	db_VPC
Subnet	Name -AZ3 CIDR – 10.0.128.0/18 Auto-assign IP = Yes
Network Interface	Eth0

6.3 Jenkins

Jenkins 2.6 will be deployed as the latest version. Jenkins requires a Master and Slave dedicated machine with will be a Docker container but on dedicated hosts. This integrates it directly with the CI method.

The CI pipeline will be kicked off by a commit to a GitHub repository. The commit will cause Jenkins to run a build job inside a Docker container, and, upon successful completion of that job, push a Docker image up to Amazon Container Registry (ECR).

Attribute	Specification
Role	Jenkins Master and Slave
Name	Jenkins-01 / Jenkins-02
Instance Details	Instance Type = t2.medium vCPU = 2 Memory = 4GB Network Performance = Low - Moderate Instance Storage = EBS only
VPC	vdmVPC

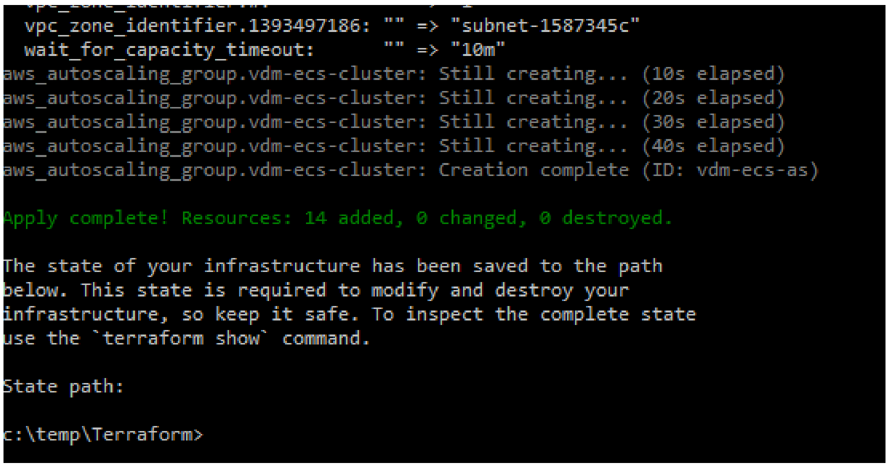
7 Operational Guide

7.1 Using Terraform to deploy Docker image on Amazon ECS

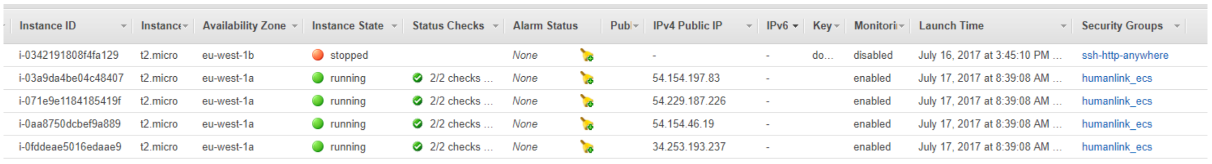
The following is a script to use with Terraform to connect to the newly formed datacentre and deploy the require infrastructure and deploys an Amazon EC2 Container Service cluster using an image with the latest Docker installed. The script will also run a task to start up a container running a basic website from the Docker hub to show as an example, the script can then be changed for the HumanLink application. The script includes the following.

- Region - eu-west-1
- VPC – vdmVPC / 200.0.0.0/16
- Internet gateway – vdmIG
- Public subnet – vdmPubSN0
- AZ – eu-west-1a
- Security Groups – vdm_load_balancers / humanlink_ecs
- ECS cluster – vdm-ecs
- Auto scaling group – min =1 max =5 desired =4
- Instance type = t2.micro
- Container tasks - 4 running
- Container image - training/webapp:latest

Run from any machine with the correct AWS access keys and secret keys, the script should report as complete.

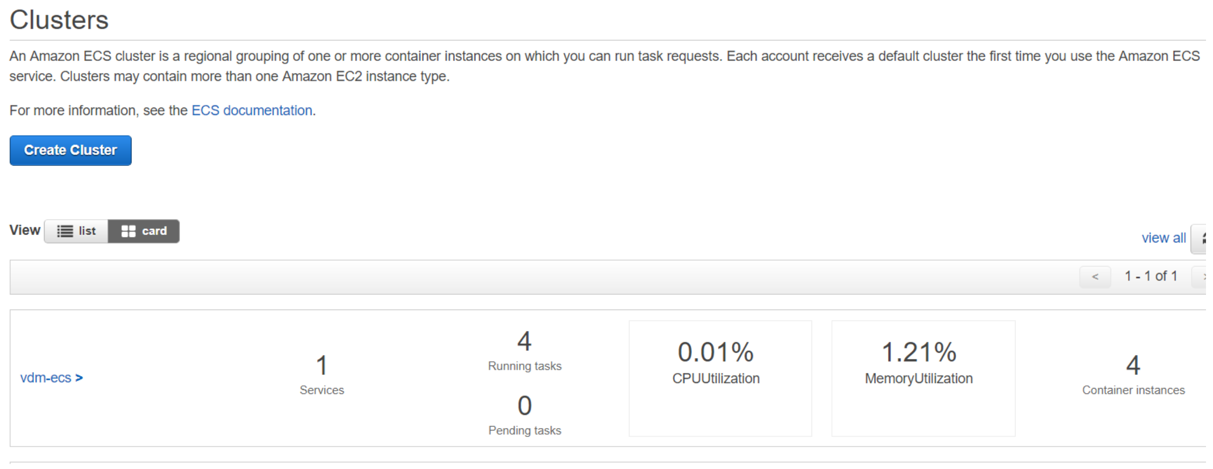

```
  vpc_zone_identifier.1393497186: "" => "subnet-1587345c"
  wait_for_capacity_timeout:      "" => "10m"
aws_autoscaling_group.vdm-ecs-cluster: Still creating... (10s elapsed)
aws_autoscaling_group.vdm-ecs-cluster: Still creating... (20s elapsed)
aws_autoscaling_group.vdm-ecs-cluster: Still creating... (30s elapsed)
aws_autoscaling_group.vdm-ecs-cluster: Still creating... (40s elapsed)
aws_autoscaling_group.vdm-ecs-cluster: Creation complete (ID: vdm-ecs-as)

Apply complete! Resources: 14 added, 0 changed, 0 destroyed.

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path:

c:\temp\Terraform>
```

Once complete 4 new instances will be created



Instance ID	Instance	Availability Zone	Instance State	Status Checks	Alarm Status	Publ	IPv4 Public IP	IPv6	Key	Monitori	Launch Time	Security Groups
i-0342191808f4fa129	t2.micro	eu-west-1b	● stopped		None		-	-	do...	disabled	July 16, 2017 at 3:45:10 PM ...	ssh-http-anywhere
i-03a9da4be04c48407	t2.micro	eu-west-1a	● running	✓ 2/2 checks ...	None		54.154.197.83	-		enabled	July 17, 2017 at 8:39:08 AM ...	humanlink_ecs
i-071e9e1184185419f	t2.micro	eu-west-1a	● running	✓ 2/2 checks ...	None		54.229.187.226	-		enabled	July 17, 2017 at 8:39:08 AM ...	humanlink_ecs
i-0aa8750dcbef9a889	t2.micro	eu-west-1a	● running	✓ 2/2 checks ...	None		54.154.46.19	-		enabled	July 17, 2017 at 8:39:08 AM ...	humanlink_ecs
i-0fddeae5016edaae9	t2.micro	eu-west-1a	● running	✓ 2/2 checks ...	None		34.253.193.237	-		enabled	July 17, 2017 at 8:39:08 AM ...	humanlink_ecs

Note the new ECS cluster created, once the instances are online they will be added to the cluster.

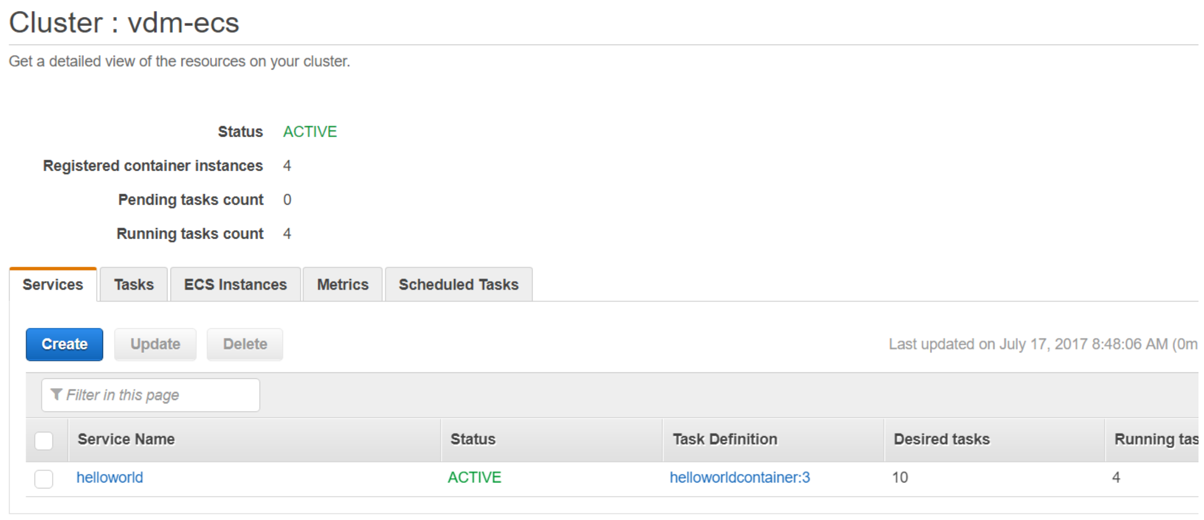


Clusters

An Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests. Each account receives a default cluster the first time you use the Amazon ECS service. Clusters may contain more than one Amazon EC2 instance type.

For more information, see the ECS documentation.

vdm-ecs >	1 Services	4 Running tasks / 0 Pending tasks	0.01% CPUUtilization	1.21% MemoryUtilization	4 Container instances



Cluster : vdm-ecs

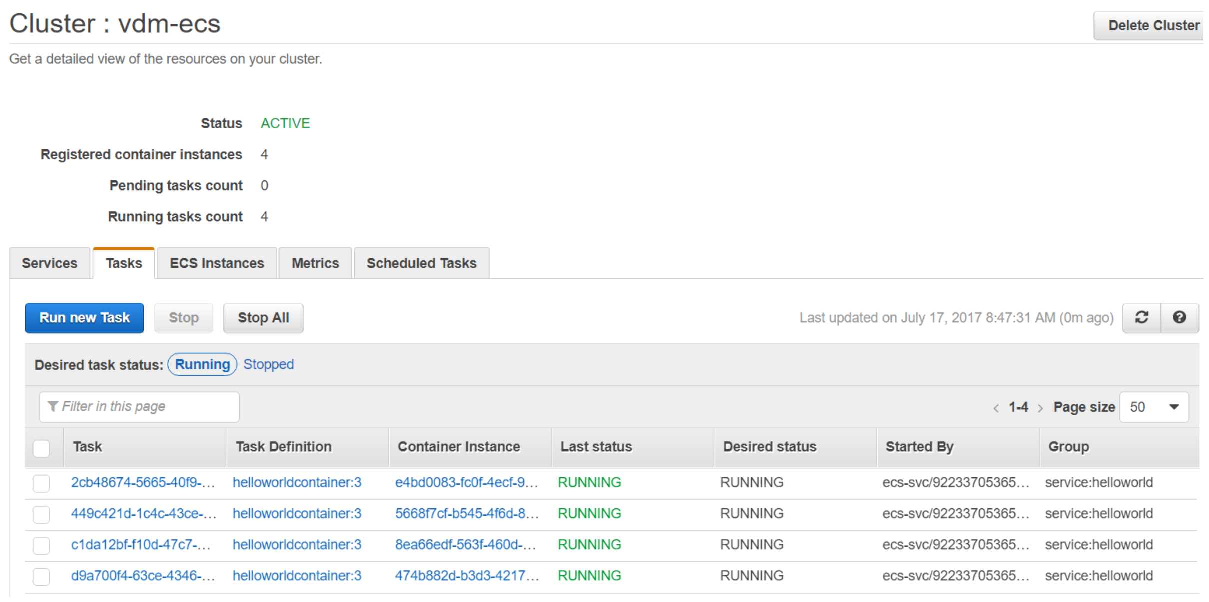
Get a detailed view of the resources on your cluster.

Status	ACTIVE
Registered container instances	4
Pending tasks count	0
Running tasks count	4

Last updated on July 17, 2017 8:48:06 AM (0m

Service Name	Status	Task Definition	Desired tasks	Running tas
helloworld	ACTIVE	helloworldcontainer:3	10	4

Once the instances are running the tasks will start.



Cluster : vdm-ecs

Get a detailed view of the resources on your cluster.

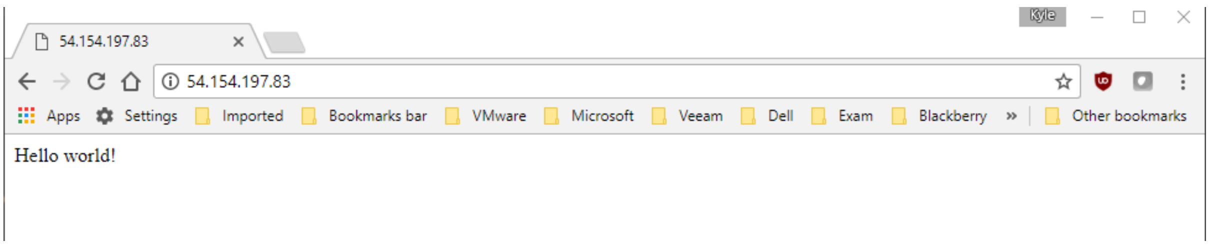
Status	ACTIVE
Registered container instances	4
Pending tasks count	0
Running tasks count	4

Last updated on July 17, 2017 8:47:31 AM (0m ago)

Desired task status: Running Stopped

Task	Task Definition	Container Instance	Last status	Desired status	Started By	Group
2cb48674-5665-40f9-…	helloworldcontainer:3	e4bd0083-fc0f-4ecf-9…	RUNNING	RUNNING	ecs-svc/92233705365…	service:helloworld
449c421d-1c4c-43ce-…	helloworldcontainer:3	5668f7cf-b545-4f6d-8…	RUNNING	RUNNING	ecs-svc/92233705365…	service:helloworld
c1da12bf-f10d-47c7-…	helloworldcontainer:3	8ea66edf-563f-460d-…	RUNNING	RUNNING	ecs-svc/92233705365…	service:helloworld
d9a700f4-63ce-4346-…	helloworldcontainer:3	474b882d-b3d3-4217…	RUNNING	RUNNING	ecs-svc/92233705365…	service:helloworld

Once the task has started the website is publicly accessible.



Hello world!

Below is the script.

Please note – variable "amis" must reflect the created image with the latest Docker installed, the below script is using .

```
# Variables
variable "region" {
    description = "Used for ECS launch control."
    default = "eu-west-1"
}

variable "amis" {
    description = "Which AMI to spawn. Defaults to the AWS ECS optimized images."
    default = {
        eu-west-1 = "ami-809f84e6"
    }
}

# Configure AWS Provider
provider "aws" {
  region     = "eu-west-1"
  access_key = ""
  secret_key = ""
}

# Define a vpc
resource "aws_vpc" "vdmVPC" {
  cidr_block = "200.0.0.0/16"
  tags {
    Name = "vdmVPC"
  }
}

# Internet gateway for the public subnet
```

```
resource "aws_internet_gateway" "vdmIG" {
  vpc_id = "${aws_vpc.vdmVPC.id}"
  tags {
    Name = "ecsvdmIG"
  }
}

# Public subnet
resource "aws_subnet" "vdmPubSN0-0" {
  vpc_id = "${aws_vpc.vdmVPC.id}"
  cidr_block = "200.0.0.0/24"
  availability_zone = "eu-west-1a"
  tags {
    Name = "ecsvdmPubSN0-0-0"
  }
}

# Routing table for public subnet
resource "aws_route_table" "vdmPubSN0-0RT" {
  vpc_id = "${aws_vpc.vdmVPC.id}"
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = "${aws_internet_gateway.vdmIG.id}"
  }
  tags {
    Name = "vdmPubSN0-0RT"
  }
}

# Associate the routing table to public subnet
resource "aws_route_table_association" "vdmPubSN0-0RTAssn" {
  subnet_id = "${aws_subnet.vdmPubSN0-0.id}"
  route_table_id = "${aws_route_table.vdmPubSN0-0RT.id}"
}

resource "aws_security_group" "vdm_load_balancers" {
    name = "vdm_load_balancers"
    description = "Allows all traffic"
    vpc_id = "${aws_vpc.vdmVPC.id}"
    # configure ports
    ingress {
        from_port = 0
        to_port = 65535
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    # configure ports.
    egress {
        from_port = 0
        to_port = 65535
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }
}

resource "aws_security_group" "humanlink_ecs" {
    name = "humanlink_ecs"
    description = "Allows all traffic"
    vpc_id = "${aws_vpc.vdmVPC.id}"

    ingress {
        from_port = 0
        to_port = 65535
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    ingress {
        from_port = 0
        to_port = 65535
        protocol = "tcp"
        security_groups = ["${aws_security_group.vdm_load_balancers.id}"]
    }

    egress {
        from_port = 0
        to_port = 65535
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]

    }

}

resource "aws_ecs_cluster" "vdm-main" {

    name = "vdm-ecs"

}


resource "aws_autoscaling_group" "vdm-ecs-cluster" {

    availability_zones = ["eu-west-1a"]

    name = "vdm-ecs-as"

    min_size = 2

    max_size = 5

    desired_capacity = 4

    health_check_type = "EC2"

    launch_configuration = "${aws_launch_configuration.vdm-ecs-lc.name}"

    vpc_zone_identifier = ["${aws_subnet.vdmPubSN0-0.id}"]

}


resource "aws_launch_configuration" "vdm-ecs-lc" {

    name = "esc-lc"

    image_id = "${lookup(var.amis, var.region)}"

    instance_type = "t2.micro"

    security_groups = ["${aws_security_group.humanlink_ecs.id}"]

    iam_instance_profile = "${aws_iam_instance_profile.ecs_ip.name}"

    associate_public_ip_address = true

    user_data = "#!/bin/bash\necho ECS_CLUSTER=vdm-ecs > /etc/ecs/ecs.config"

}


resource "aws_iam_instance_profile" "ecs_ip" {

    name = "ecs-instance-profile"

    roles = ["${aws_iam_role.ecs_instance_role.name}"]

}


resource "aws_iam_role" "ecs_instance_role" {

    name = "ecs-instance-role"

    path = "/"

    assume_role_policy = <<EOF

{

  "Version": "2008-10-17",

  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
}

resource "aws_iam_role_policy" "ecs_instance_role" {
    name = "ecs-instance-role"
    role = "${aws_iam_role.ecs_instance_role.id}"
    policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecs:CreateCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:DiscoverPollEndpoint",
        "ecs:Poll",
        "ecs:RegisterContainerInstance",
        "ecs:StartTelemetrySession",
        "ecs:Submit*"
      ],
      "Resource": "*"
    }
  ]
}
EOF
}

resource "aws_iam_role_policy" "ecs_scheduler_role" {
    name = "ecs-scheduler-role"
    role = "${aws_iam_role.ecs_instance_role.id}"
    policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:Describe*",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
        "elasticloadbalancing:Describe*",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer"
      ],
      "Resource": "*"
    }
  ]
}
EOF
}

data "aws_ecs_task_definition" "helloworld" {
  task_definition = "${aws_ecs_task_definition.helloworldcontainer.family}"
}

resource "aws_ecs_task_definition" "helloworldcontainer" {
  family = "helloworldcontainer"

  container_definitions = <<DEFINITION
[
  {
    "cpu": 128,
    "portMappings": [
      {
        "containerPort": 5000,
        "hostPort": 80
      }
    ],
    "essential": true,
    "image": "training/webapp:latest",
    "memory": 128,
    "memoryReservation": 64,
    "name": "helloworld"
  }
]
DEFINITION
}

resource "aws_ecs_service" "helloworld" {
  name          = "helloworld"
  cluster       = "${aws_ecs_cluster.vdm-main.id}"
  desired_count = 4
  task_definition = "helloworldcontainer"

}
```

8 References

Netflix Cassandra case study - https://medium.com/netflix-techblog/benchmarking-cassandra-scalability-on-aws-over-a-million-writes-per-second-39f45f066c9e

University of Toronto Cassandra case study - http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf

Docker / Jenkins - https://goto.docker.com/rs/929-FJL-178/images/20150825-continuous-integration-pipeline.pdf?mkt_tok=eyJpIjoiT0RGbU9HWXhNbUk0TnpCbCIsInQiOiJ6ajFcL0xcL212blwvRmVOUndCSUVFT2F5UnNETHdUc1ZYMzhkaTJQWXE1RTFkd1NoUHZpOENKS1RZQzdWTWJYNkxHanZrTUFhektCY2pua041dWFjbklhQjFhMGZuckVFeUVpMklKRmdFcGtDSE5aRFwvblVLYTNUT2lkR0l3K1dcL1I3In0%3D

Amazon ECR - https://aws.amazon.com/ecr/details/

Docker on AWS - http://www.ybrikman.com/writing/2015/11/11/running-docker-aws-ground-up/

Terraform ECS task - https://serverfault.com/questions/843498/terraform-how-do-i-have-1-ecs-cluster-with-2-or-more-ecs-service-task-definition

ECS task definitions - http://docs.aws.amazon.com/AmazonECS/latest/developerguide/example_task_definitions.html

Cassandra on EC2 - https://d0.awsstatic.com/whitepapers/Cassandra_on_AWS.pdf

Terraform - https://blog.gruntwork.io/a-comprehensive-guide-to-terraform-b3d32832baca

Jenkin Pipelines - https://jenkins.io/2.0/#pipelines

Jenkins with Terraform - https://objectpartners.com/2016/06/01/automating-terraform-projects-with-jenkins/