# ECS Quickstart Documentation



# Authors

Everton Evaristo

[everte@amazon.com](mailto:everte@amazon.com)

Revision by:

## Confidentiality of Contents

The contents of this document are shared only under NDA and should not be reattributed.

## Amazon Web Services Disclaimer

This document is not legally, binding, and is not an offer to contract that can be accepted by either party. All responses,

descriptions and observations in this document are informational and are provided solely for discussion purposes. Neither

party will have any obligation or liability with respect to the matters described in this document. All obligations must be

set forth in a separate definitive agreement executed by the parties addressing such matters, provided, however, that

neither party will have any liability for any failure or refusal to enter into a definitive agreement for any reason. Amazon

Web Services, Inc. (AWS) has provided responses based on its current knowledge, but these responses may change at any

time due to a variety of factors, including without limitation, changes to your requirements, the capabilities of any third

party you select to assist with implementation, and changes to AWS’s service offerings. AWS does not make any

representations or warranties of any kind in this document. Any use of the AWS service offerings will be governed by the

AWS Customer Agreement available at <http://aws.amazon.com/agreement/>(or other definitive written agreement

between the parties), not this document. AWS does not accept any terms or conditions included in this document that

conflict with or are in addition to the terms and conditions set forth in the AWS Customer Agreement.

# Overview

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster. Your containers are defined in a task definition that you use to run individual tasks or tasks within a service. In this context, a service is a configuration that enables you to run and maintain a specified number of tasks simultaneously in a cluster.

You can run your tasks and services on a serverless infrastructure that is managed by AWS Fargate. Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon EC2 instances that you manage.

Amazon ECS enables you to launch and stop your container-based applications by using simple API calls. You can also retrieve the state of your cluster from a centralised service and have access to many familiar Amazon EC2 features.

You can schedule the placement of your containers across your cluster based on your resource needs, isolation policies, and availability requirements. With Amazon ECS, you don't have to operate your own cluster management and configuration management systems or worry about scaling your management infrastructure.

This Quick Start is for Developer professionals who want to use the AWS ECS service managed by Fargate, meaning that we run our containers in serverless infrastructure.. This QuickStart provides the step by step guide to create and deploy an AWS ECS Cluster, Task Definitions, an alternative AWS RDS Mysql  database and CloudWatch Logs and the necessary operational alarms.

# Architecture

This Quick Start implements a ECS cluster in the Itaú's accounts using the DevOps model, this also deploy the Services and Task definitions to run workloads in this environment. As an optional feature, deploy one Amazon RDS database as part of the stack. This base architecture uses 3 pipelines in the *tools* account.

The infrastructure uses Terraform in the background, and the application stack uses AWS Cloudformation. This stacks uses FARGATE as backend for running containers. FARGATE it's a technology who runs containers without the need of *fleets*provisioned instances. It's a *serverless* architecture designed to run containers.

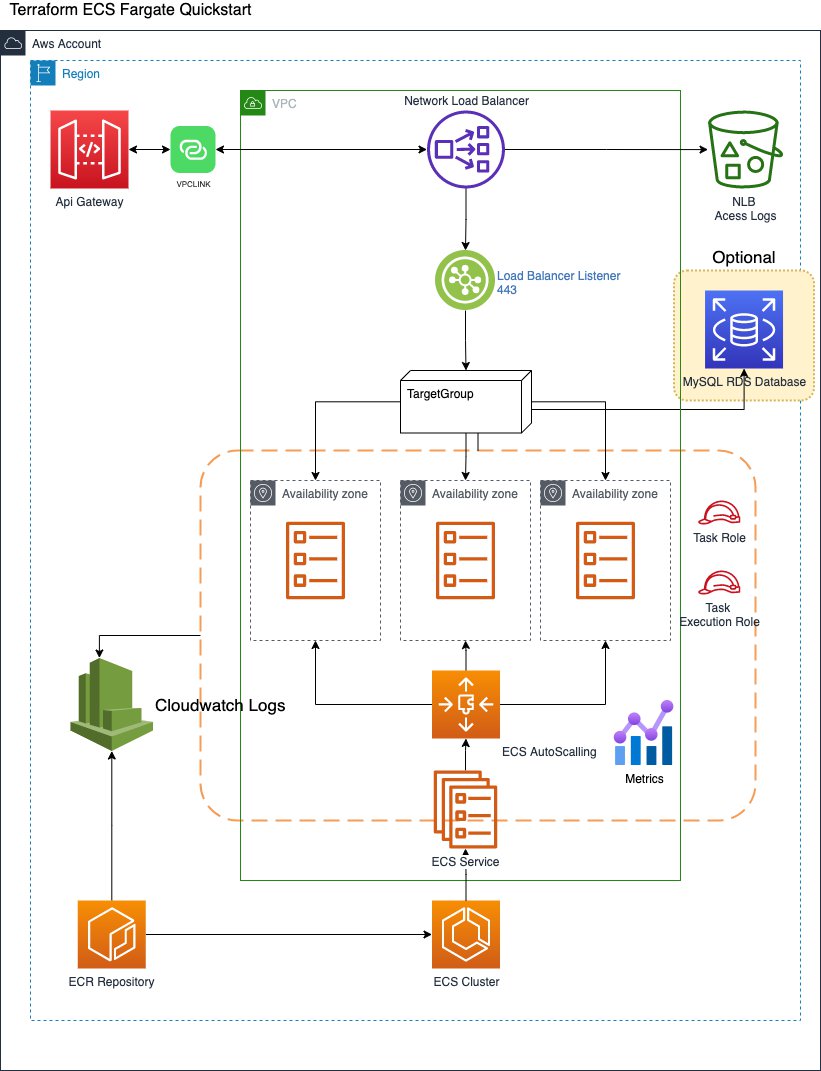


fig1. High Level ECS Architecture

### Infrastructure

The ECS stack deploy's the cluster itself, the Network Load Balancer and the S3 bucket to keep the access logs from the NLB. Below are the arguments need by the execution of this pipeline:

# General Parameters  
region = "us-east-1"  
app = "itau-ecs-cluster"  
environment = "dev"  
tags = { "application" : "bankappecsfargate", "environment" : "dev", "team" : "Cloud Engineering", "contact-email" : "contact@amazon.com", "customer" : "itau" }

Where:

|  |  |
| --- | --- |
| Column 1 | Column 2 |
| **Parameter** | **Description** |
| region | Region where the cluster will be created |
| app | Name of the cluster |
| environment | The name of the environment of creation |
| tags | Tags used in the resource |

### Application

#Todo

### RDS (Optional)

#Todo

## Amazon API Gateway Integration

Itaú uses Amazon API Gateway as a gateway for any client located outside of the service's network perimeter that needs to connect with web services running in the application's accounts.

As defined by the "Integração Digital" team, each web service exposed through the API Gateway should have a dedicated Network Load Balancer, and a VPC Link created. This stack creates both, the NLB (Network Load Balancer) and the VPC Link as exposed in the fig1.

## Splunk Integration

Todo

## Prometheus and Grafana Integration

Todo

## Design Controls

This Quickstart implements the following controls defined by Itaú:

Security Controls:

* Encryption of data at rest and in flight.

Logging \ Monitoring:

* Enable ECS Cluster Container Insights logging to CloudWatch logs.
* Integration of Prometheus & Grafana.
* Integration into existing Splunk account.

Automation:

* Automated provisioning of ECS Clusters through CI\CD pipelines.
* Automated provisioning ECS Tasks through CI\CD pipelines.

Scalability:

* Ability to scale the quantity of task in and out to satisfy workload requirements.

Networking

* Network Load Balancers (NLB) will be utilised to balance traffic across application pods within the ECS cluster.

# Planning the Deployment

## Specialized Knowledge

This deployment requires a moderate level of familiarity with AWS services. If you’re new to AWS, visit Getting Started with AWS and Training and Certification. These sites provide materials for learning how to design, deploy, and operate your infrastructure and applications on the AWS Cloud.

This Quick Start also assumes that you are familiar with Terraform, AWS CodePipeline, AWS Codebuild, ECR, AWSCLI, IAM, and AWS ECS and container technologies.

## AWS Account

If you don’t already have an AWS account, create one at http://cloud.itau by following the instructions in the cloud journey.

You are charged only for the services you use.

## Technical Requirements

Before you launch the Quick Start, your account must be configured as specified in the following table. Otherwise, deployment might fail.

## Supported regions

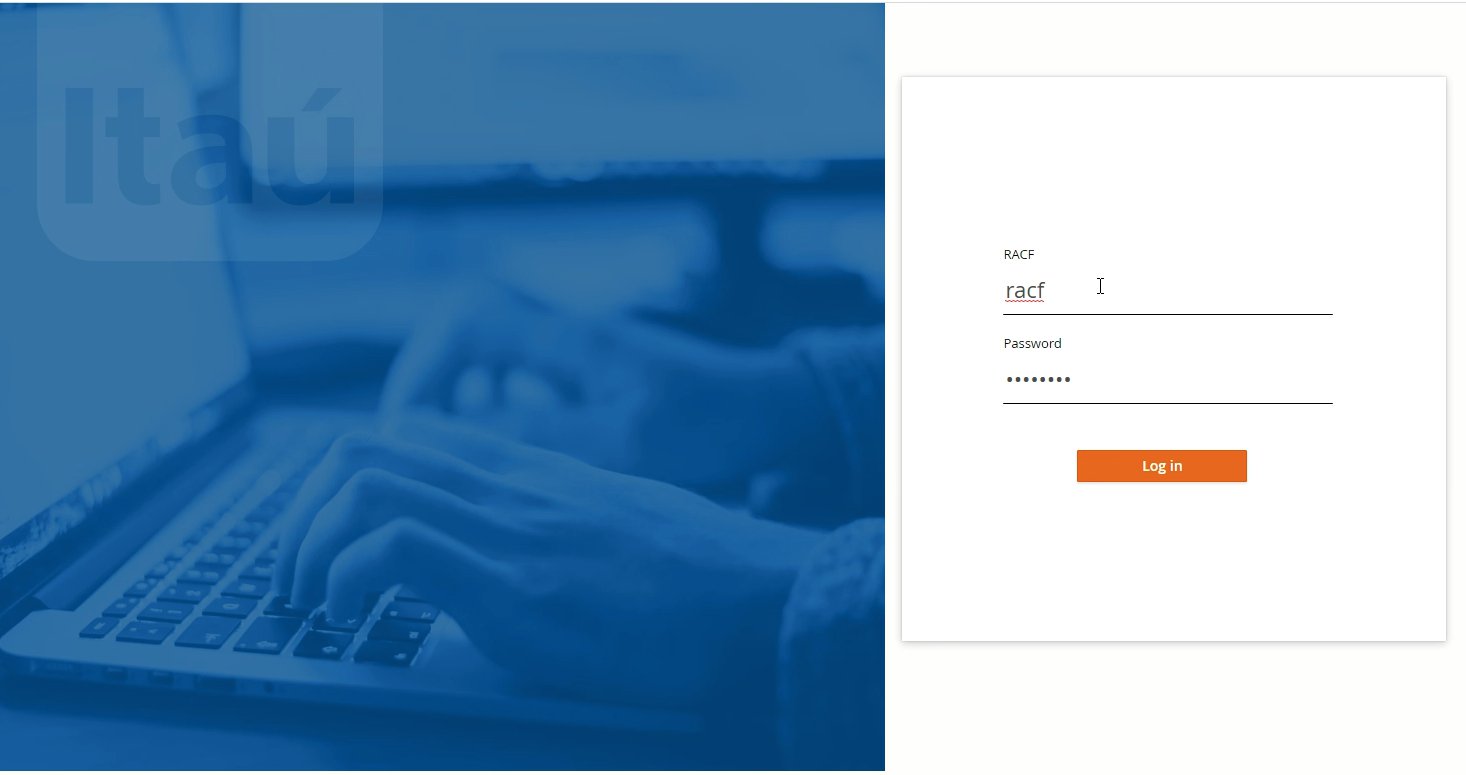
This Quick Start supports the following Regions:

* sa-east-1, South America

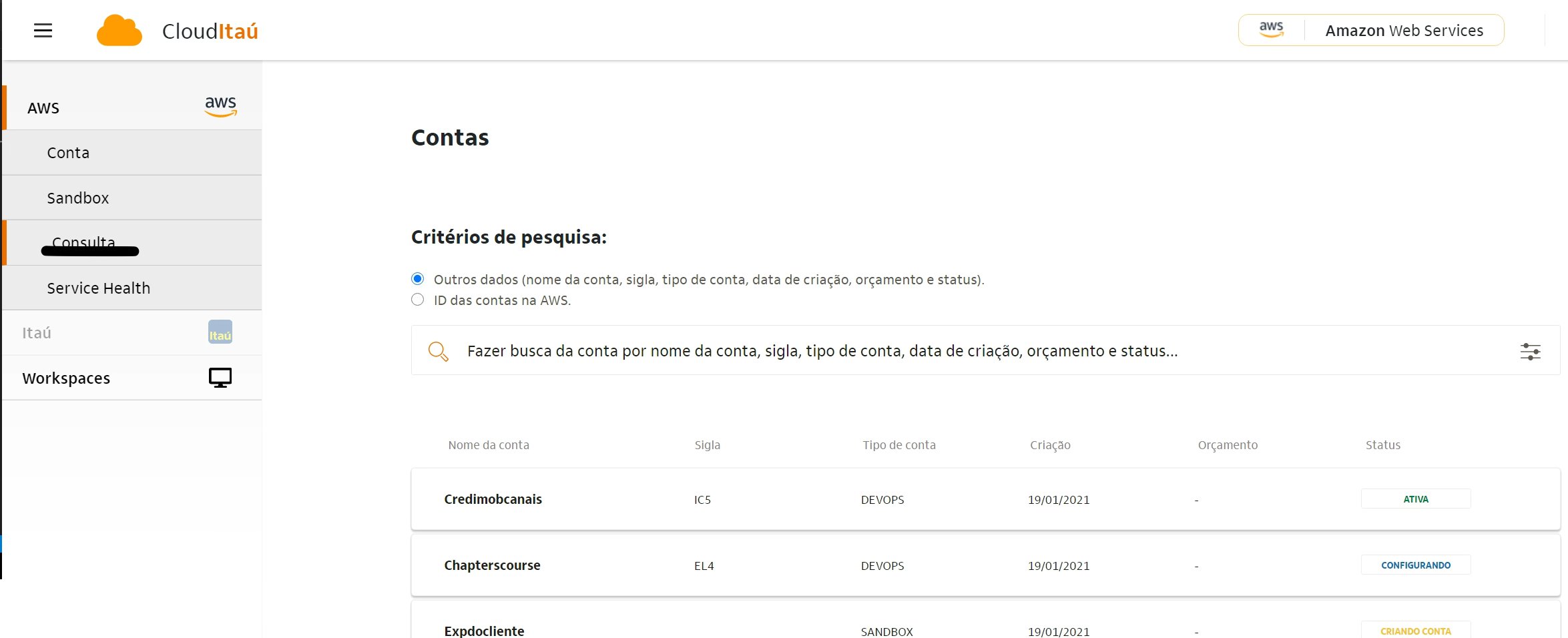
## CI/CD pipeline

According to the DevOps model defined by Itaú's, after having an AWS account ready, you should create an infrastructure pipeline for deploying an AWS Step Functions State Machine as part of your application. For that, please follow the procedure describe below:

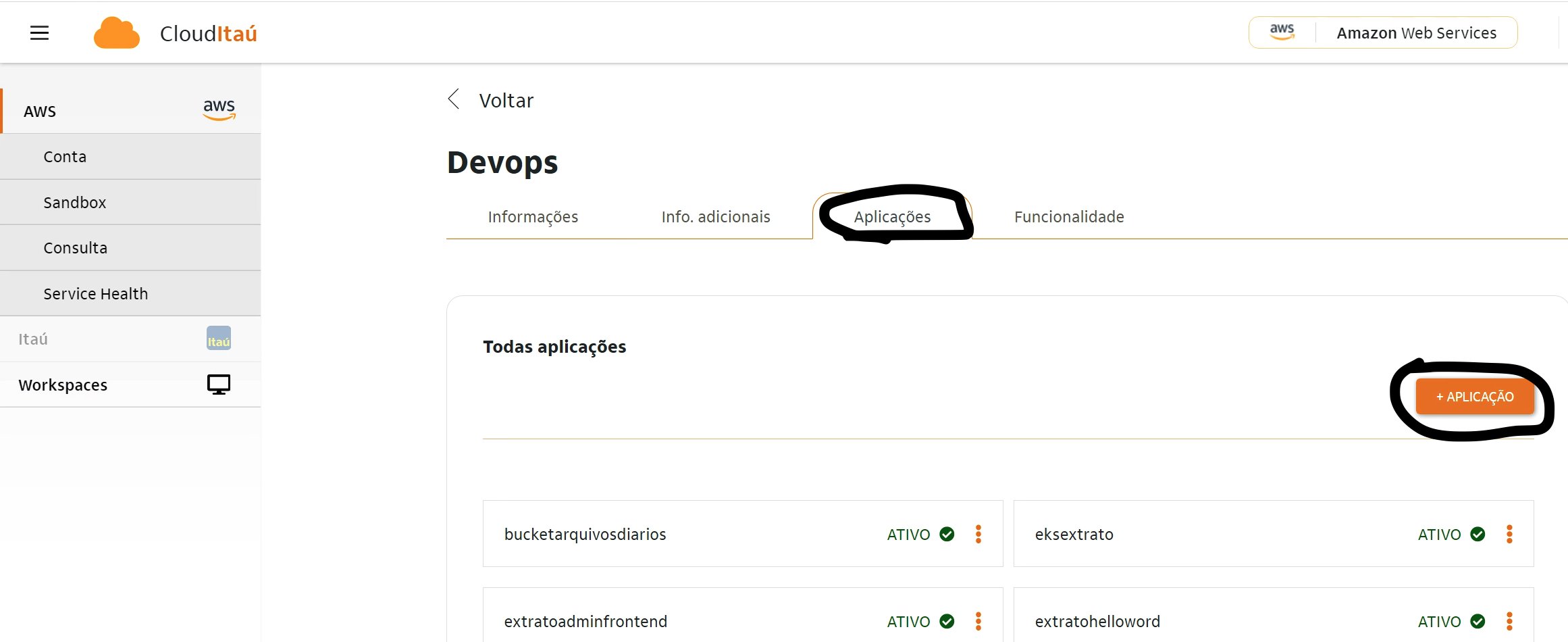
1. Log into the [Itaú Portal](https://cloud-itau.cloud.ihf/)



1. Search for your AWS account using the “Consulta” option at the left



1. After finding your account go to the “Aplicaçoes” (applications) tab and create an application. **Please use Terraform as the Runtime Infrastructure as Code (IaC).**



1. After a successful creation you should be able to search your pipeline (application) which should be in an “ative” (active) state and contain the information about the GitLab repository created.

### IAM Permissions

To deploy this Quick Start to your account, the DevOps role used by the CI/CD pipeline must have permission to create and delete the resources used by this Quick Start. The current pipeline doesn't have enough permissions to execute. So it's necessary to include the following:

The following permissions are need for Terraform who use s3 as backend for state files

* [s3:ListBucket](https://www.terraform.io/docs/language/settings/backends/s3.html#s3-listbucket) on arn:aws:s3:::mybucket
* [s3:GetObject](https://www.terraform.io/docs/language/settings/backends/s3.html#s3-getobject) on arn:aws:s3:::mybucket/path/to/my/key
* [s3:PutObject](https://www.terraform.io/docs/language/settings/backends/s3.html#s3-putobject) on arn:aws:s3:::mybucket/path/to/my/key

ref: <https://www.terraform.io/docs/language/settings/backends/s3.html>

The following permission are need for Terraform creates a NetworkLoadBalancer

* [elasticloadbalancing:\*](https://www.terraform.io/docs/language/settings/backends/s3.html#s3-listbucket)

## Supported Regions

It can be deployed to sa-east-1. IAM roles and VC info in the variables files will need to be updated.

# Deployment Steps

Current steps are a code push to the develop branch on the repository.

A codebuild pipeline is triggered from this push and builds to the develop environment.

A merge to the master branch deploys this to hom environment and is then built to prod.

There is a variables file located within the repo which can be modified to make changes or update the cluster config per environment, including by not limited to, the type and count of the instances in the cluster.

As mentioned before, there are 3 pipelines, lets start by the Infrastructure, who will create the NLB and ECS Cluster

## Infrastructure Stack

## **Dev Environment**

### Step 1 - Clone the Quick start repo from Gitlabs

Clone the repo [get oficial repo] . This repo is still a work in progress.

### Step 2 - Edit the variables file

Located in infra/inventories/{environment}/terraform.tfvars

|  |  |
| --- | --- |
| Column 1 | Column 2 |
| **Parameter** | **Description** |
| region | Region where the cluster will be created |
| app | Name of the cluster |
| environment | The name of the environment of creation |
| tags | Tags used in the resource |
| private\_subnets | Id's of the private subnets |
| public\_subnets | Id's of the public subnets |
| vpc | Vpc-Id of the the account |
| lb\_port | NLB Port |
| lb\_protocol | NLB Protocol |
| internal | NLB Schema (Internal or Internet Facing) |
| deregistration\_delay | Time to de-register a target |
| health\_check | Health Check path |
| health\_check\_interval | Health Check interval |
| health\_check\_timeout | Health Check Timeout |
| lb\_access\_logs\_expiration\_days | Log retention days |

### Step 3 - Push the code to the develop branch

Code pushed to the Develop branch, this get synced with the CodeCommit Repo and activates the CodebuildPipeline.

### Step 4 - Watch the CI/CD pipeline

Check the CI/CD pipeline and fix any error detected during the deployment of the Terraform.

## **Hom/Prod Environment**

### Step 1 - Edit the variables file

Edit the file located at infra/inventories/{hom/prod}/terraform.tfvars and modify accordingly.

### Step 2 - Push the code to the master branch

* Create a merge request to the master branch, using the “Entrege de historias tecnicas” option in the Description drop down.
  + Replace the “CHAVES DA HISTORIA, CHAVES DA HISTORIA” text within the bracket with the JIRA history ticket number.  
    ie. [ CCCTENABL-4400 ]
* When the build kicks off and gets to the  “Governanca” secrtion, a lambda is fired to pair the JIRA history ticket with a deployment ticket.
  + This ticket can be found by clicking the (i) in the CriaCardJira section of the pipeline, clicking the link to the fired lambda, and finding the CARD\_KEY Variable associated to it in JIRA, ie CCENTABL-5325
  + Once you find the ticket in JIRA, assign the ticket to yourself and click the “Preform Planning" button to schedule a deployment time for the prod deployment. Keep in mind theres another approval step so account for this time.
  + Once this step has been completed a ticket is opened for a Manager to approve the deployment. Reach out to Bruno Parus Boassi or Danilo Franca Silva for now, to complete this deployment.

### Step 4 - Watch the CI/CD pipeline

Check the CI/CD pipeline and fix any error detected during the deployment of the Terraform.

## Application Stack

Todo

## Database Stack

Todo

# Post Deployment Steps

Todo

# Additional Resources

## Send Containers logs to CloudWatch Logs

### Introduction

It's possible to sent container logs to cloudwatch, you will need to setup the fluentbit deamonset on cluster.

### Prerequisites

* Create an IAM role and service account (IRSA) with allowed policy to create log streams on Cloudwatch

### Setup

In the following steps, you set up Fluent Bit as a daemonSet to send logs to CloudWatch Logs. When you complete this step, Fluent Bit creates the following log groups if they don't already exist.

|  |  |  |
| --- | --- | --- |
|  | Log Group Name | Log Source |
| 1 | /aws/containerinsights/Cluster\_Name/application | All log files in /var/log/containers |
| 2 | /aws/containerinsights/Cluster\_Name/host | Logs from /var/log/dmesg, /var/log/secure, and /var/log/messages |
| 3 | /aws/containerinsights/Cluster\_Name/dataplane | The logs in /var/log/journal for kubelet.service, kubeproxy.service, and docker.service. |

**Step 1 - Create Namespace**

kubectl create namespace amazon-cloudwatch

**Step 2 - Create ConfigMap**

Sample fo Spec config file

apiVersion: v1  
kind: ConfigMap  
metadata:  
   name: fluent-bit-cluster-info  
   namespace: amazon-cloudwatch  
data:  
  cluster.name: <CLUSTER NAME>  
  http.server: "On"  
  http.port: "2020"  
  read.head: "On"  
  read.tail: "On"  
  logs.region: <REGION NAME>

**Step 3 - Apply this spec**

kubectl -n amazon-cloudwatch -f file.yaml

**Step 4 - Apply Deployment, svc and others specs**

kubectl apply -f <https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-bit/fluent-bit-compatible.yaml>

## AWS LoadBalancer Controller

### Introduction

This feature enables ingress in LoadBalancer new services.

(<https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html>)

### Prerequisites

1 - You need to create a LoadBalancer (NLB or ALB)

2 - Create a IAM Role with grant policy to change LoadBalancers and conditional access (see IAM Roles for Service Accounts (IRSA))

### Step 1 - Install AWS loadBalancer Controller

Download the controller specification. For more information about the controller, see the [documentation](https://kubernetes-sigs.github.io/aws-load-balancer-controller/) on GitHub.

curl -o v2\_1\_2\_full.yaml https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.1.2/docs/install/v2\_1\_2\_full.yaml

Make the following edits to the 2\_1\_2\_full.yaml file:

Delete the ServiceAccount section from the specification. Doing so prevents the annotation with the IAM role from being overwritten when the controller is deployed and preserves the service account that you created in step 4 if you delete the controller.

Set the --cluster-name value to your Amazon EKS cluster name in the Deployment spec section.

Apply the file.

kubectl apply -f v2\_1\_2\_full.yaml

### Step 2 - Validate deployment

$ kubectl get pods  
$ kubectl logs <LB Controller pod name>

### Step 3 - Deploy a node port service and TargetGroupBinding

When you deploy a new service on cluster you need to deploy a spec with TargetGroupBinding to ingress this service on LoadBalancer. Above a example of Spec File

apiVersion: v1  
kind: Service  
metadata:  
   name: <service-name>  
spec:  
   type: NodePort #Service Expose  
   ports:  
      - port: 30036 #Port expose  
      name: http #Port Name  
      targetPort: 8080 #Service Port  
      nodePort: 30036  
   selector:  
      app:<deployment name>  
---  
apiVersion: elbv2.k8s.aws/v1beta1  
kind: TargetGroupBinding  
metadata:  
   name: <tgb-name>  
spec:  
   targetType: instance  
   serviceRef:  
      name: <service-name>  
      port: <service-port>  
      targetGroupARN: <Target-Group-ARN>

The TargetGroupBinding spec values

    openAPIV3Schema:  
      description: TargetGroupBinding is the Schema for the TargetGroupBinding API  
      properties:  
        apiVersion:  
          description: 'APIVersion defines the versioned schema of this representation  
            of an object. Servers should convert recognized schemas to the latest  
            internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources'  
          type: string  
        kind:  
          description: 'Kind is a string value representing the REST resource this  
            object represents. Servers may infer this from the endpoint the client  
            submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds'  
          type: string  
        metadata:  
          type: object  
        spec:  
          description: TargetGroupBindingSpec defines the desired state of TargetGroupBinding  
          properties:  
            networking:  
              description: networking provides the networking setup for ELBV2 LoadBalancer  
                to access targets in TargetGroup.  
              properties:  
                ingress:  
                  description: List of ingress rules to allow ELBV2 LoadBalancer to  
                    access targets in TargetGroup.  
                  items:  
                    properties:  
                      from:  
                        description: List of peers which should be able to access  
                          the targets in TargetGroup. At least one NetworkingPeer  
                          should be specified.  
                        items:  
                          description: NetworkingPeer defines the source/destination  
                            peer for networking rules.  
                          properties:  
                            ipBlock:  
                              description: IPBlock defines an IPBlock peer. If specified,  
                                none of the other fields can be set.  
                              properties:  
                                cidr:  
                                  description: CIDR is the network CIDR. Both IPV4  
                                    or IPV6 CIDR are accepted.  
                                  type: string  
                              required:  
                                - cidr  
                              type: object  
                            securityGroup:  
                              description: SecurityGroup defines a SecurityGroup peer.  
                                If specified, none of the other fields can be set.  
                              properties:  
                                groupID:  
                                  description: GroupID is the EC2 SecurityGroupID.  
                                  type: string  
                              required:  
                                - groupID  
                              type: object  
                          type: object  
                        type: array  
                      ports:  
                        description: List of ports which should be made accessible  
                          on the targets in TargetGroup. If ports is empty or unspecified,  
                          it defaults to all ports with TCP.  
                        items:  
                          properties:  
                            port:  
                              anyOf:  
                                - type: integer  
                                - type: string  
                              description: The port which traffic must match. When  
                                NodePort endpoints(instance TargetType) is used, this  
                                must be a numerical port. When Port endpoints(ip TargetType)  
                                is used, this can be either numerical or named port  
                                on pods. if port is unspecified, it defaults to all  
                                ports.  
                              x-kubernetes-int-or-string: true  
                            protocol:  
                              description: The protocol which traffic must match.  
                                If protocol is unspecified, it defaults to TCP.  
                              enum:  
                                - TCP  
                                - UDP  
                              type: string  
                          type: object  
                        type: array  
                    required:  
                      - from  
                      - ports  
                    type: object  
                  type: array  
              type: object  
            serviceRef:  
              description: serviceRef is a reference to a Kubernetes Service and ServicePort.  
              properties:  
                name:  
                  description: Name is the name of the Service.  
                  type: string  
                port:  
                  anyOf:  
                    - type: integer  
                    - type: string  
                  description: Port is the port of the ServicePort.  
                  x-kubernetes-int-or-string: true  
              required:  
                - name  
                - port  
              type: object  
            targetGroupARN:  
              description: targetGroupARN is the Amazon Resource Name (ARN) for the  
                TargetGroup.  
              type: string  
            targetType:  
              description: targetType is the TargetType of TargetGroup. If unspecified,  
                it will be automatically inferred.  
              enum:  
                - instance  
                - ip  
              type: string  
          required:  
            - serviceRef  
            - targetGroupARN  
          type: object  
        status:  
          description: TargetGroupBindingStatus defines the observed state of TargetGroupBinding  
          properties:  
            observedGeneration:  
              description: The generation observed by the TargetGroupBinding controller.  
              format: int64  
              type: integer  
          type: object  
      type: object

### Step 4 - Apply spec files

$ kubectl -n <namespace> apply service.yaml  
$ kubectl -n <namespace> apply targetgroupbinding.yaml

### Step 5 - Validate ingress

$ kubectl get pods  
$ kubectl logs pods <ingress-controller-pod>

## Creating a Kubernetes Application

### Step 1 - Creating IAM roles for the applications

Follow the steps of IAM Roles for Service Accounts section.

### Step 2 - Creating the Kubernetes Deployment Spec

You need to configure a spec file to your application, above a sample of YAML file with deployment, service, service account and TargetGroupBinding:

apiVersion: v1  
kind: ServiceAccount  
metadata:  
  name: <NAME>  
  namespace: <NAMESPACE>  
  annotations:  
    eks.amazonaws.com/role-arn: arn:aws:iam::<ACCOUNT NUMBER>:role/<IAM ROLE NAME>  
---  
apiVersion: v1  
kind: Service  
metadata:  
   name: <service-name>  
spec:  
   type: NodePort #Service Expose  
   ports:  
      - port: 30036 #Port expose  
      name: http #Port Name  
      targetPort: 8080 #Service Port  
      nodePort: 30036  
   selector:  
      app:<deployment name>  
---  
apiVersion: elbv2.k8s.aws/v1beta1  
kind: TargetGroupBinding  
metadata:  
   name: <tgb-name>  
spec:  
   targetType: instance  
   serviceRef:  
      name: <service-name>  
      port: <service-port>  
      targetGroupARN: <Target-Group-ARN>  
  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  labels:  
    <internal kubernetes labels>  
  name: <deployment Name>  
  namespace: <namespace>  
spec:  
  replicas: <number of replicas>  
  template:  
    metadata:  
      labels:  
        <internal kubernetes labels>  
    spec:  
      containers:  
        - args:  
          image: <repository/path/image:tag> #to create you can set a mock image, CI/CD will change this value with your deployment  
          env:  
          - name: <variable name>  
            value: <value>  
          - name: <variable name>  
            value: "value"  
          livenessProbe:  
            failureThreshold: 2  
            httpGet: #validation method  
              path: /path  
              port:<port number>  
              scheme: HTTP  
            initialDelaySeconds: 30  
            timeoutSeconds: 10  
          name: <port name>  
          ports:  
            - containerPort: <container port number>  
              name: <port name>  
              protocol: TCP  
          resources:  
            limits: #resource limits to your application  
              cpu: 200m  
              memory: 500Mi  
            requests: #inicial resource to your application  
              cpu: 100m  
              memory: 200Mi  
      serviceAccountName: <service account name>

### Step 2 - Apply spec file

$ kubectl -n <namespace> apply specfile.yaml

### Step 3 - Verifying deployment and service

$ kubectl -n <namespace> get pods  
$ kubectl -n <namespace> describe pods <pod name>  
$ kubectl -n <namespace> get svc

## Prometheus Alerts

Prometheus will be deployed to collect cluster based metrics.

Prometheus AlertManager will be utilized to trigger alerts based on the following metrics and thresholds:

|  |  |  |  |
| --- | --- | --- | --- |
| Column 1 | Column 2 | Column 3 | Column 4 |
| **Alert Type** | **Threshold Trigger Value** | **Threshold Period** | **Comments** |
| CPU Utilization %, by Node | >80% | for 5 minutes |  |
| Memory Utilization %, by Node | >80% | for 5 minutes |  |
| Filesystem Disk Utilization %, by Node | >80% | for 5 minutes |  |
| Node Count | >80% | for 5 minutes | to be set at 80% of Scaling Config max capacity |
|  |  |  |  |

The following diagrams illustrate how the the metrics above will be displayed in Prometheus:

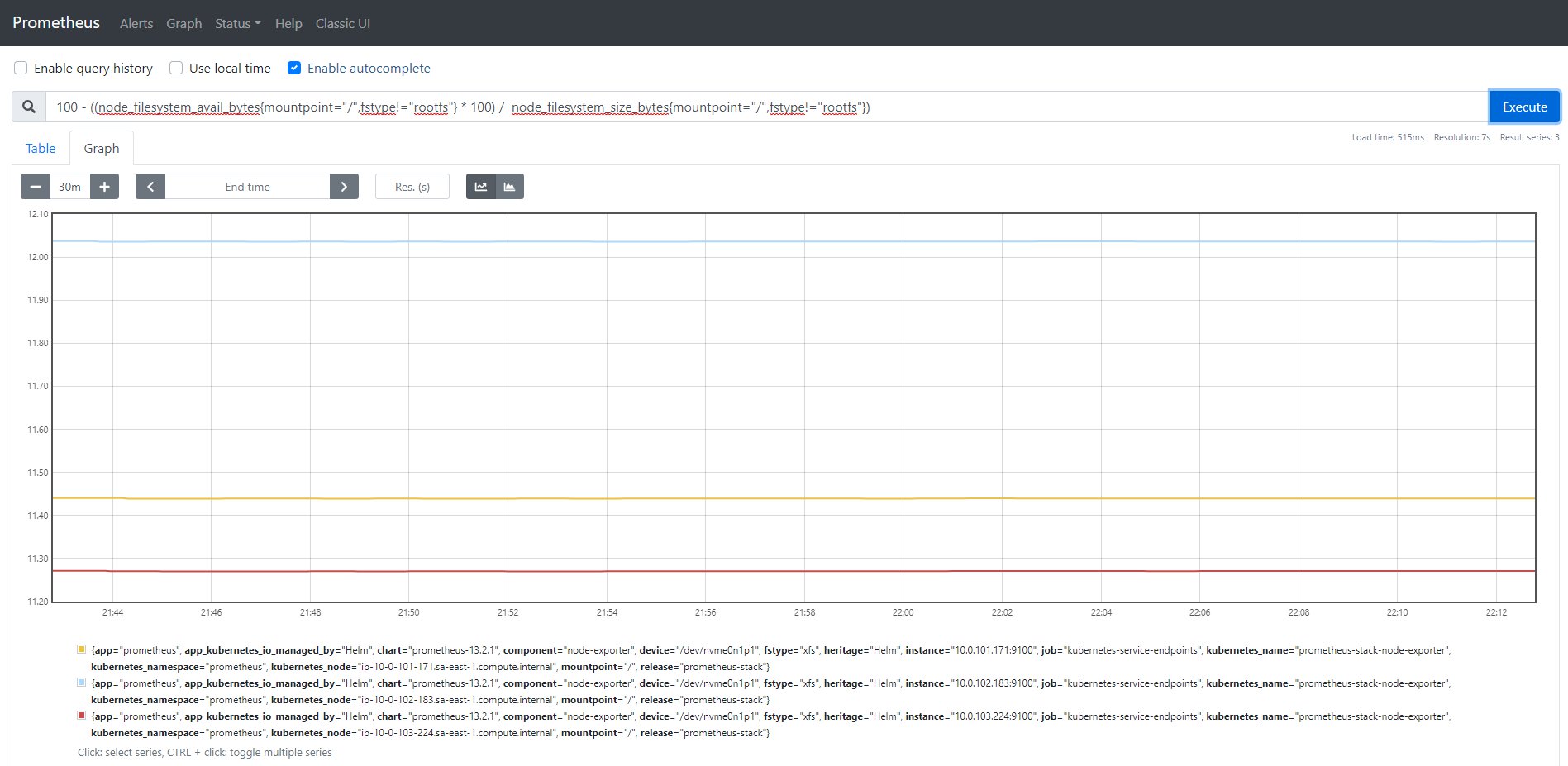
### CPU Utilization, by Node



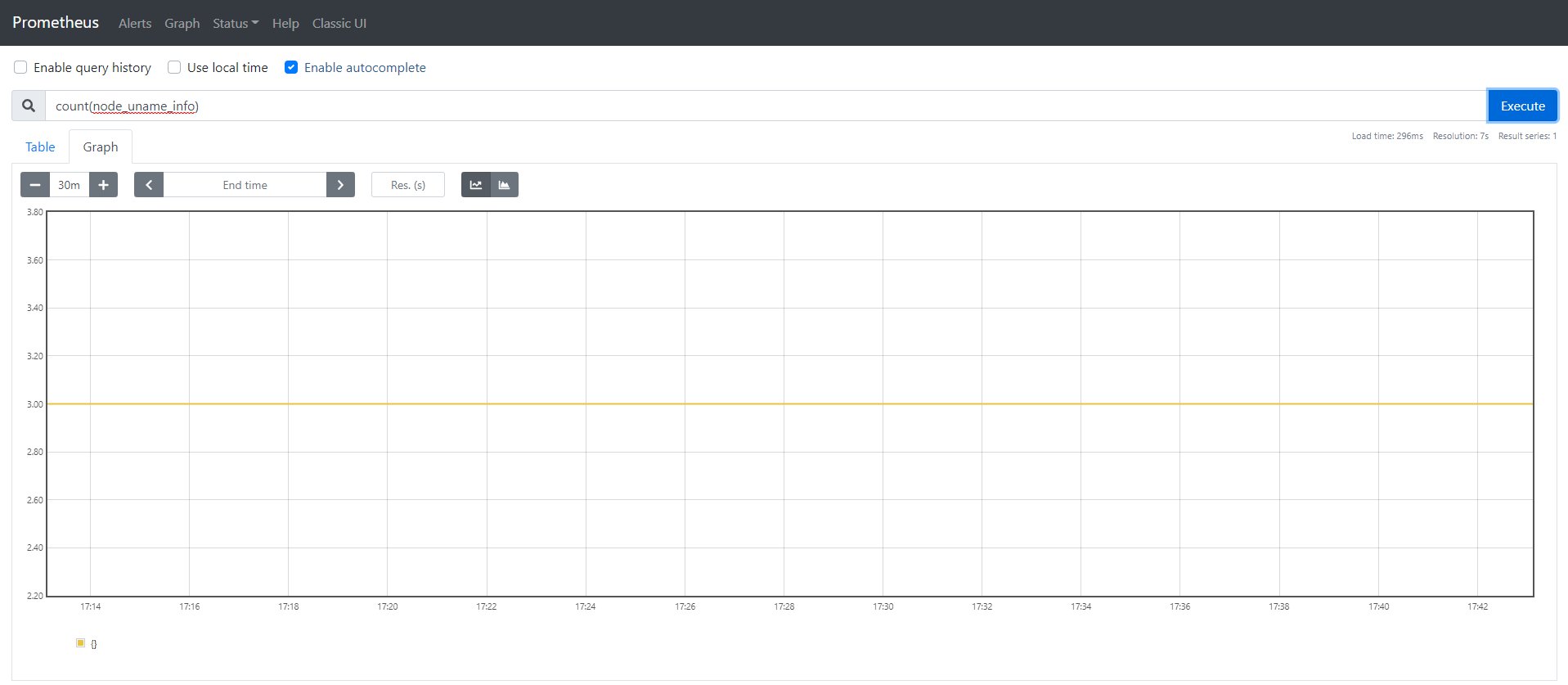
### Memory Utilization, by Node



### Filesystem Disk Utilization, by Node



### Node Count



# References

<https://www.eksworkshop.com/beginner/180_fargate/prerequisites-for-alb/>

<https://aws.amazon.com/blogs/containers/introducing-aws-load-balancer-controller/>

# FAQ

1) What operation model should I use this quickstart?

This quick start was built with the DevOps model in mind, where you have four AWS accounts: DevTools, Devolopment, Stage and Production.

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.