# How to deploy on AWS from GitLab

A single application for Lambda, Fargate, EC2, EKS, and ECS


GitLab

# What's inside?

## How can you use GitLab for AWS?

» GitLab + Lambda
» GitLab + Fargate
» GitLab + EKS
» GitLab + ECS
» GitLab + EC2

## Additional AWS functionality in the works

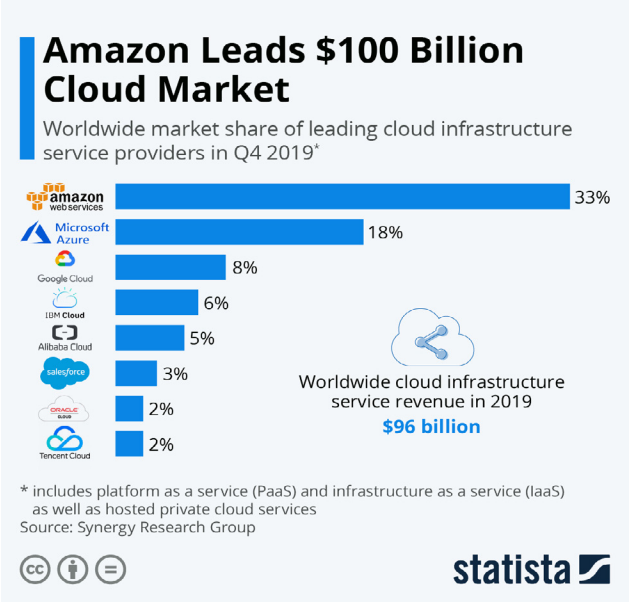
## Getting started with GitLab

» Installing GitLab from AWS Management console
» Installing GitLab from AWS Marketplace

## Benefits of using GitLab for AWS services

# Introduction

Amazon Web Services (AWS) is the de facto leader in the worldwide cloud infrastructure market, accounting for nearly 33% in 2019. As of January 2020, AWS offers nearly 160 global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security and enterprise applications. Millions of customers, including startups, large-scale enterprises, and government agencies rely on AWS cloud services to power their business.




Developer tools have a high capacity for driving cloud usage and organizations adopt cloud native application development to take advantage of the power of the cloud computing model. Teams utilize cloud native technologies to build and deploy to a limitless number of cloud services, but with small teams managing microservices written in different languages deployed using separate tools, information silos eventually emerge.

Toolchain complexity is a common thread among DevOps teams that are doing cloud native development today. In a 2019 Forrester survey of IT professionals, they found that many software development teams are struggling with managing and integrating multiple tools. In order to leverage AWS services, development teams can through five or more different tools just to deploy code. Teams then have to manage these tools. Developers end up spending a lot of time maintaining a complex toolchain rather than innovating or building new things.

Many organizations use AWS because it is an all-in-one cloud service, offering everything from storage, to networking, to serverless under one roof. AWS is meant to be comprehensive, and that can bring multiple benefits for its customers:

» **Simplicity**: All of our cloud data is one place.

» **Convenience**: AWS has everything we need.

» **Reliability**: We can rely on AWS to have solutions.

» **Ease of use**: We don't have to train our team on multiple clouds.

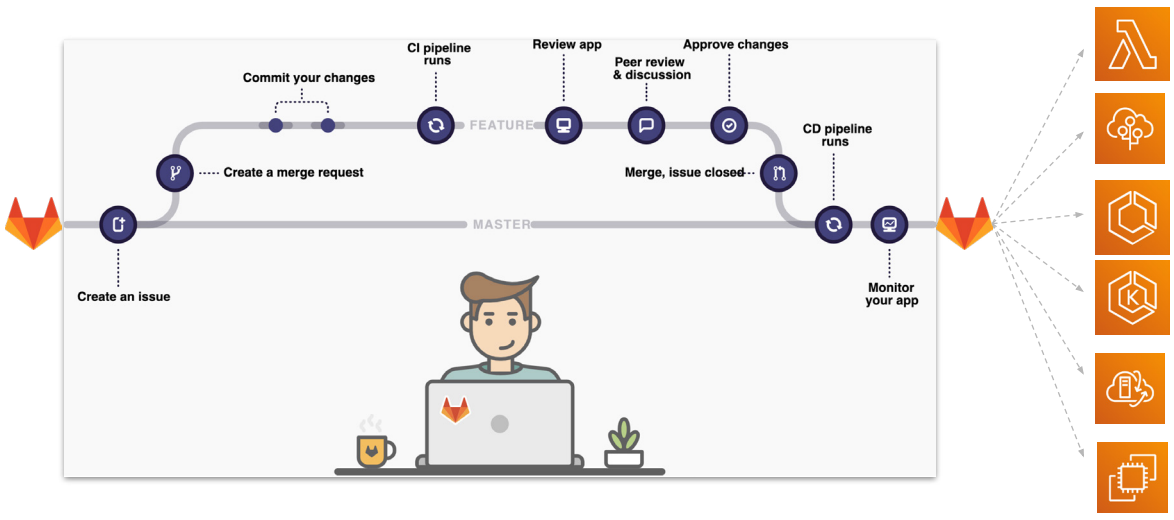
Teams that go “all-in” on AWS do so in order to simplify their cloud needs, but because AWS offers so many cloud services, this simplicity does not come at the cost of functionality.

When it comes to DevOps tools, organizations can also take advantage of this all-in-one meets functionality approach. Instead of bringing in separate tools, integrating them, and maintaining them, DevOps teams can have the entire software development lifecycle in one application with GitLab.

# How can you use GitLab for AWS?

What differentiates GitLab from other DevOps applications is that we allow teams to manage the entire software development lifecycle from one interface, eliminating the need for a complicated toolchain. Utilizing GitLab CI/CD, teams can customize deployments to any AWS service. In this white paper, we’ll provide information on how GitLab integrates with five popular AWS services: Lambda, Fargate, EKS, ECS, and EC2.

GitLab: Concurrent Development Workflow




13

# GitLab + Lambda

In the Cloud Native Computing Foundation (CNCF) 2019 survey, 41% of respondents use serverless technology. Of those using serverless, 80% use a hosted platform vs. 20% who use installable software. Of those using a hosted platform, the top tool is AWS Lambda (53%).

AWS Lambda is the leader when it comes to building serverless applications. In order to complete a serverless application, organizations will need the following:

» A computing service

» A database service

» An HTTP gateway service

Lambda is an AWS proprietary compute service that integrates with other AWS services as well as open source solutions. The majority of developers building serverless applications today rely on Lambda functions.

The biggest roadblocks to serverless adoption remain tooling and workflows. Organizations love the scalability and automation of serverless but don't believe that they have the tools to implement it effectively. Enterprises already working within a complicated toolchain are understandably hesitant to add yet another tool into their stack.

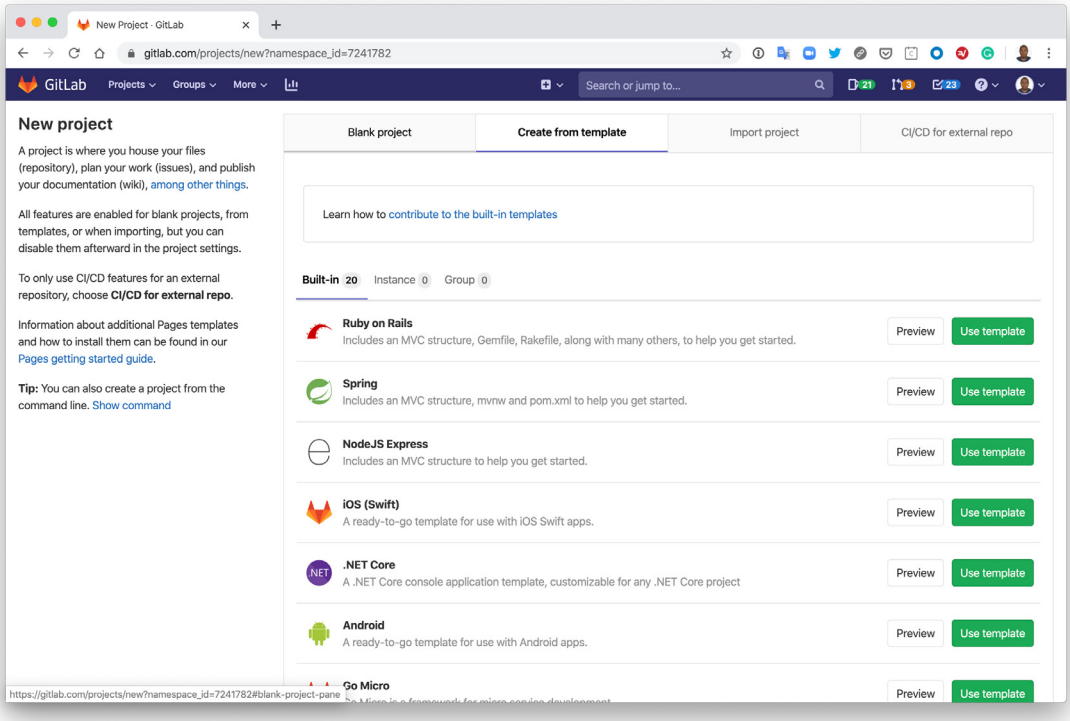
GitLab allows users to deploy AWS Lambda functions and create rich serverless applications using a combination of:

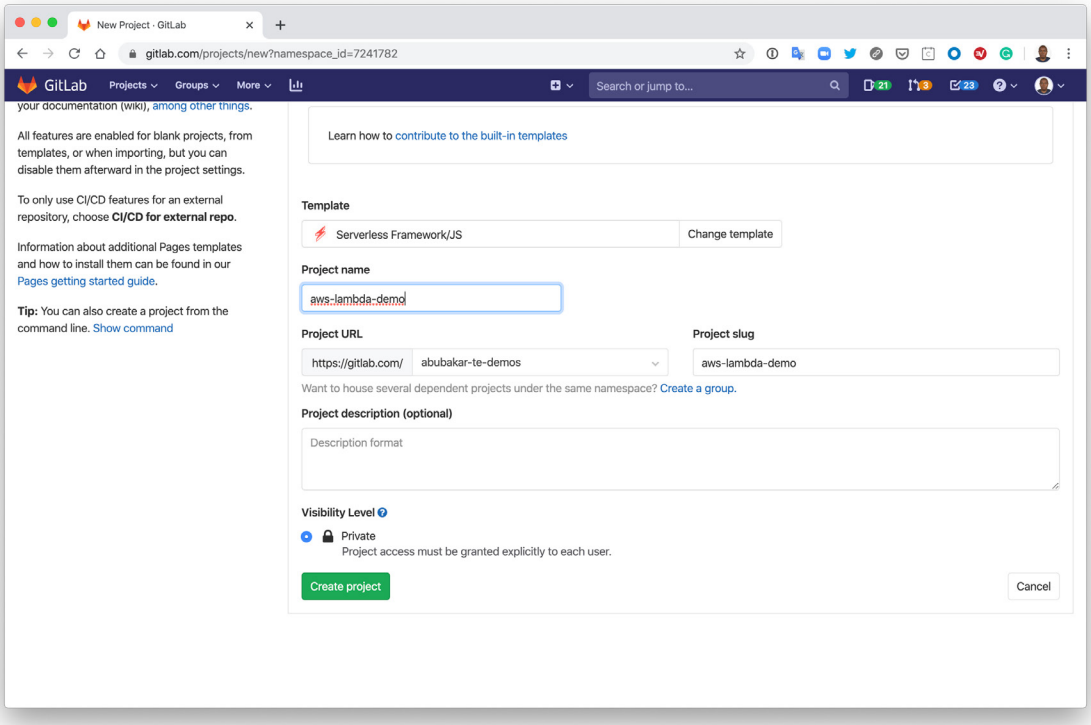
» Serverless Framework with AWS

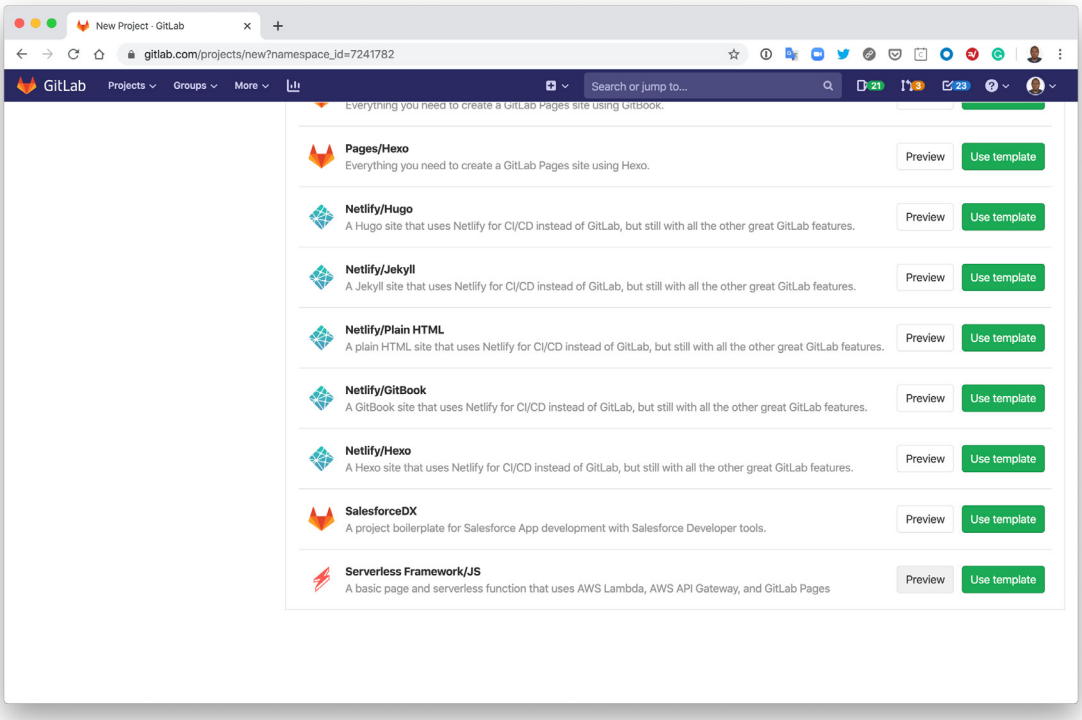
» AWS Serverless Application Model (SAM)

» GitLab CI/CD

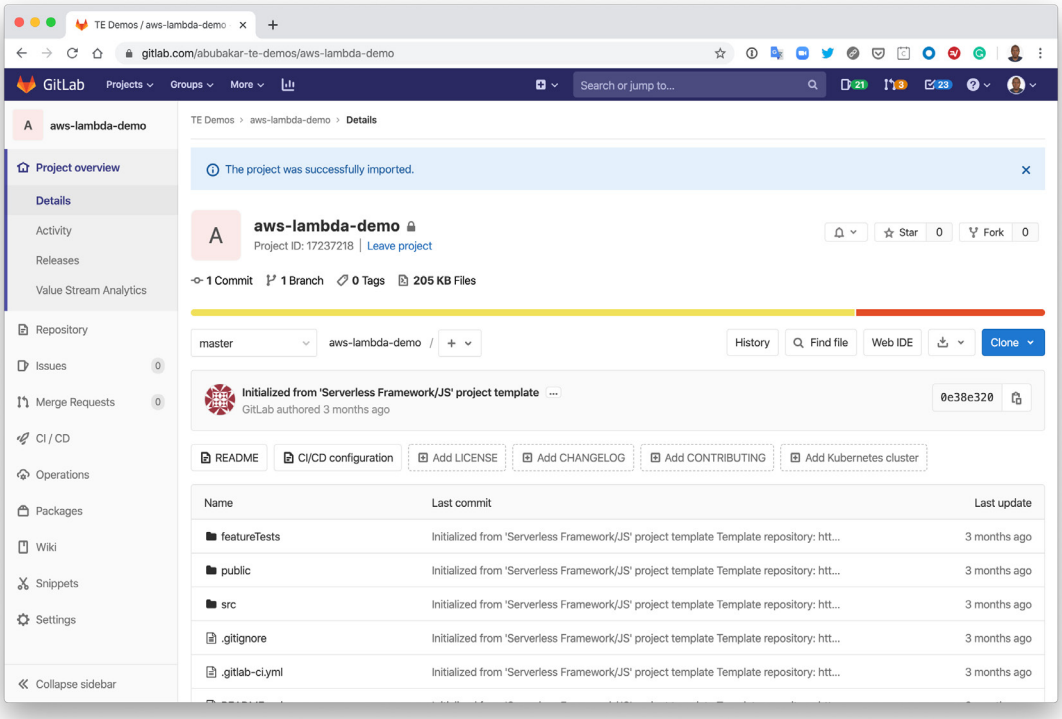
GitLab users have the option to create a project from a Serverless Framework/JS template:



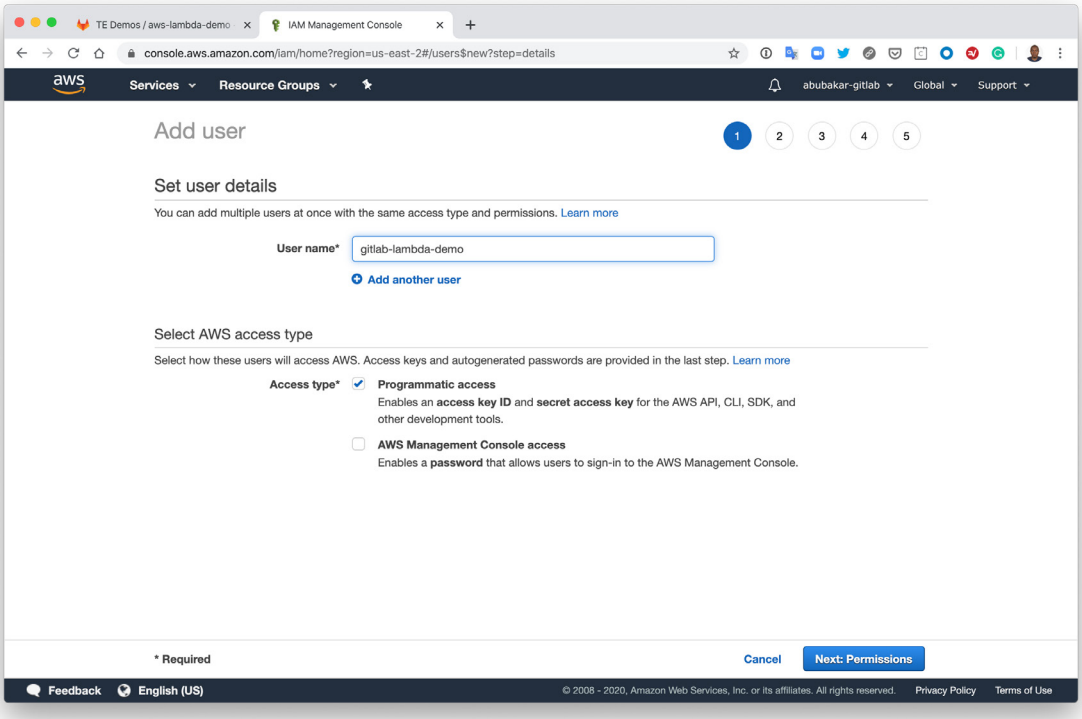





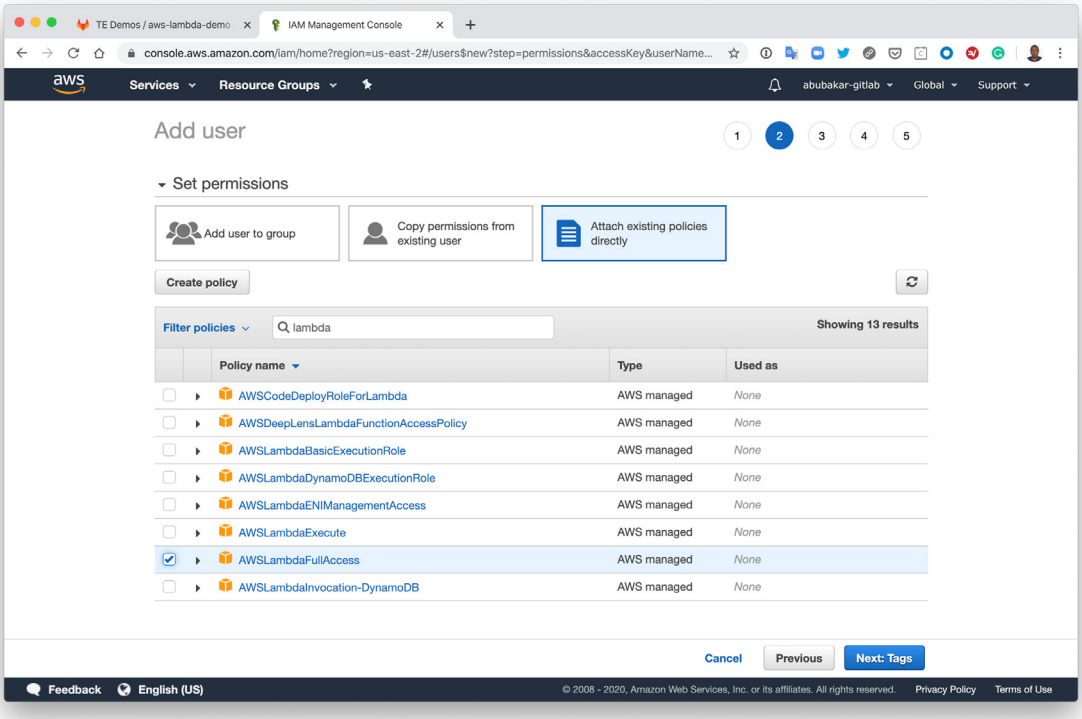

Once the import of the template is complete, you can then work with a sample project:



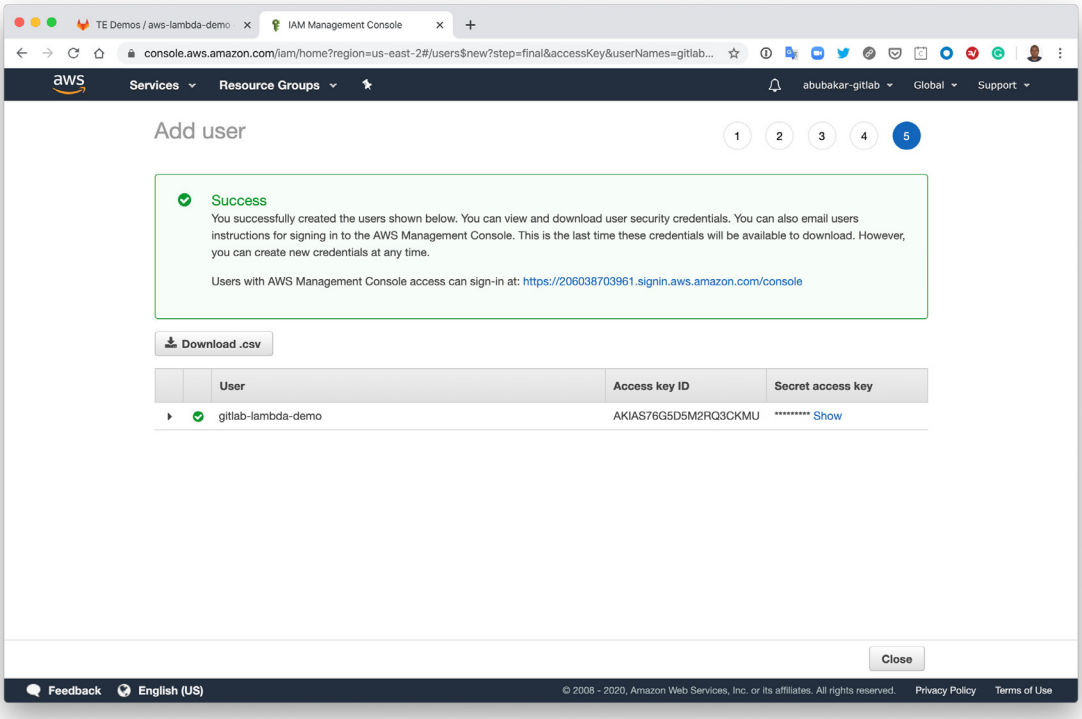
Next step: create an AWS IAM account to use for authentication:



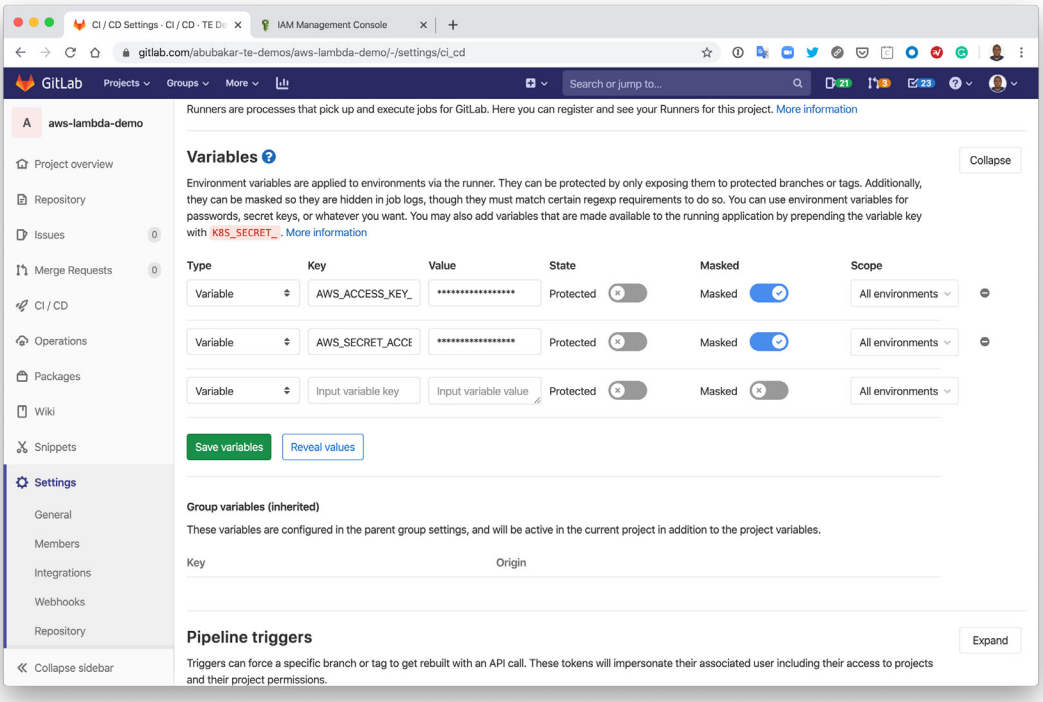

Attach the necessary policies to the user:



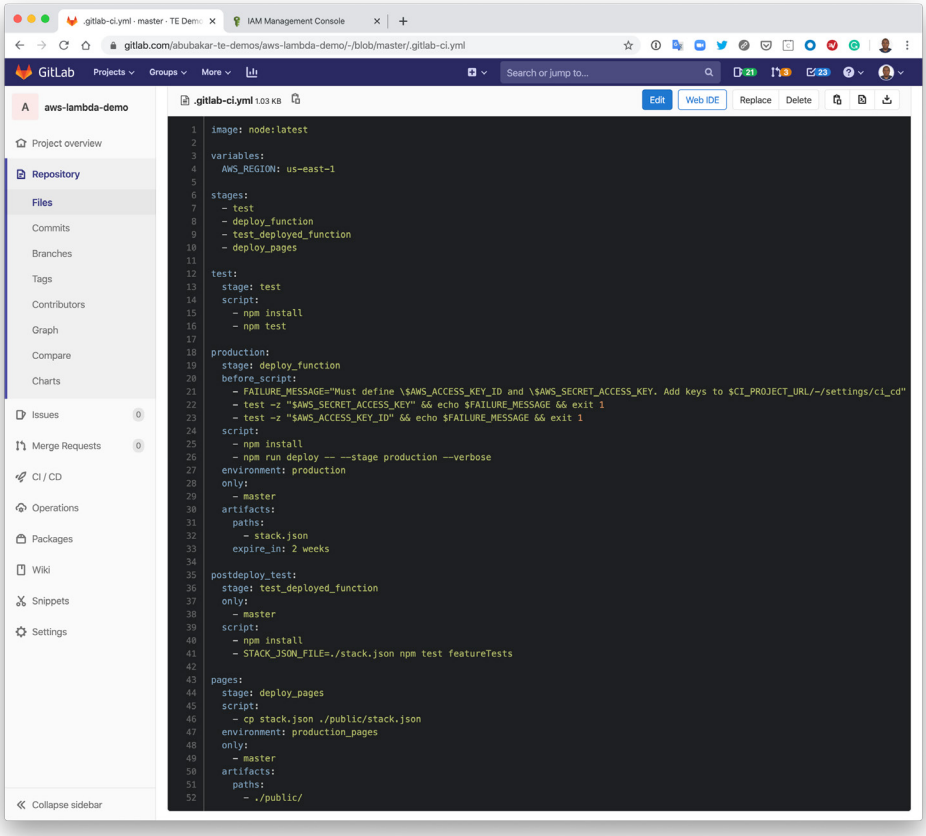
Once these steps are completed, you should have the following summary:

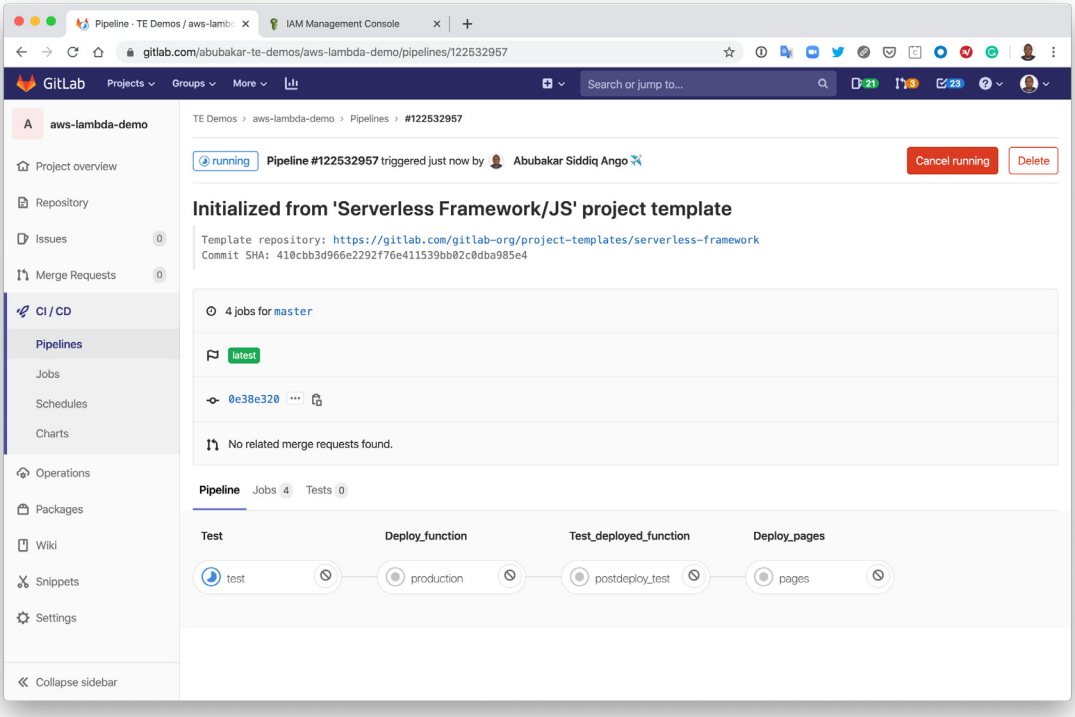


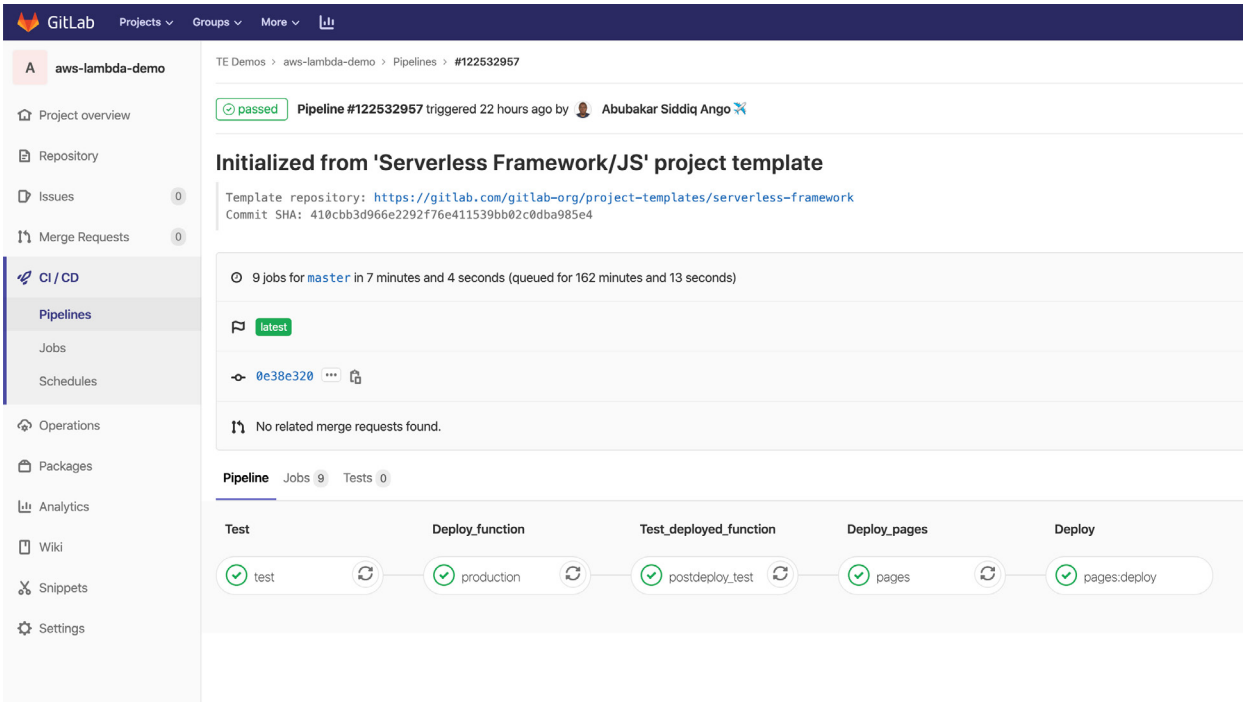

Next step: Add the AWS_ACCESS_KEY and AWS_SECRET_ACCESS_KEY as CI/CD variables to the project as shown below:



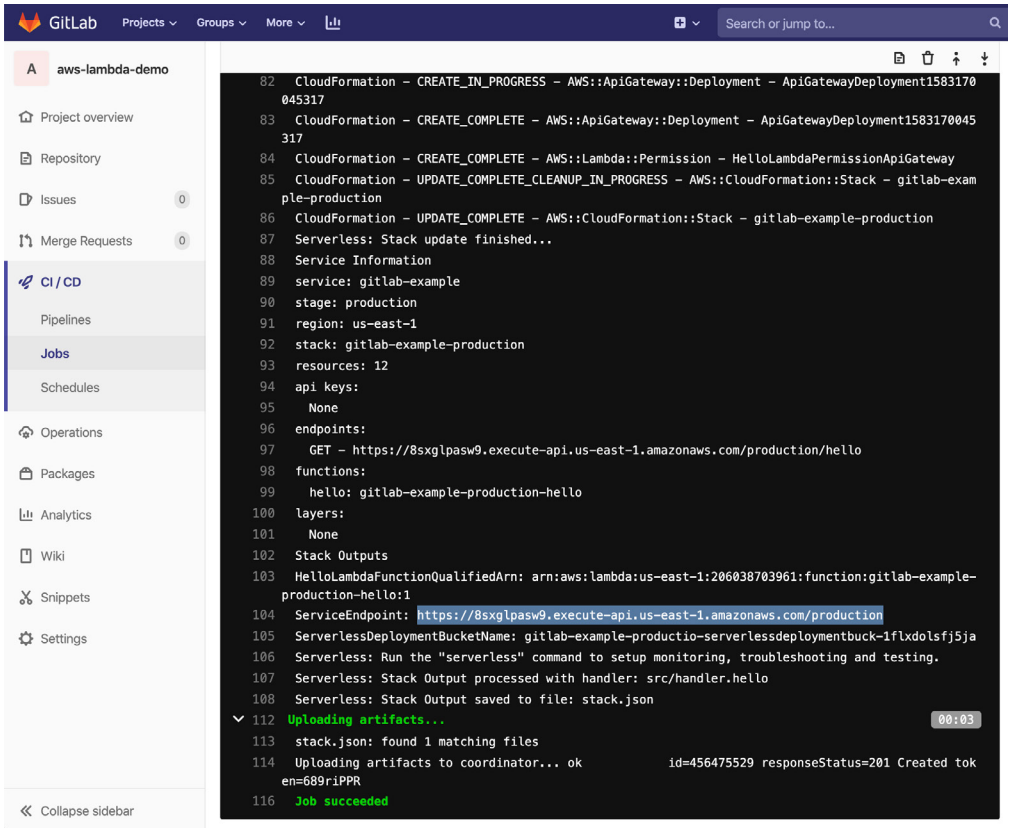
The project sample contains a .gitlab-ci.yml file. Changes made to the repository, or when a pipeline is initiated, four CI jobs (test, production, postdeploy_test and pages) are run as shown below:



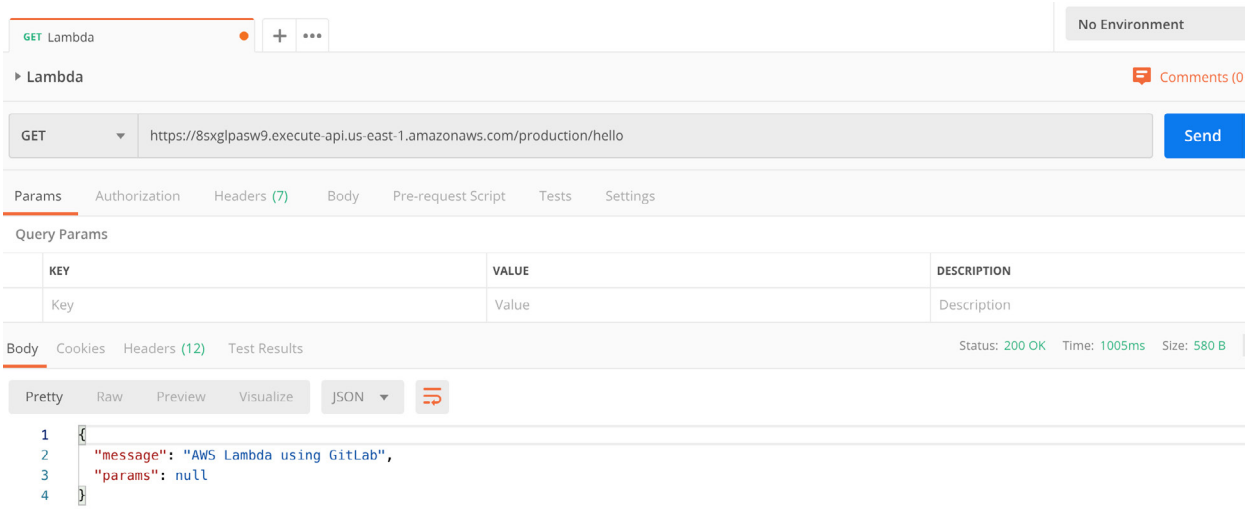






Once the production job completes, the URL of the Lambda endpoint is created:



In this example, when a GET request is sent to the URL, it returns the following sample text:



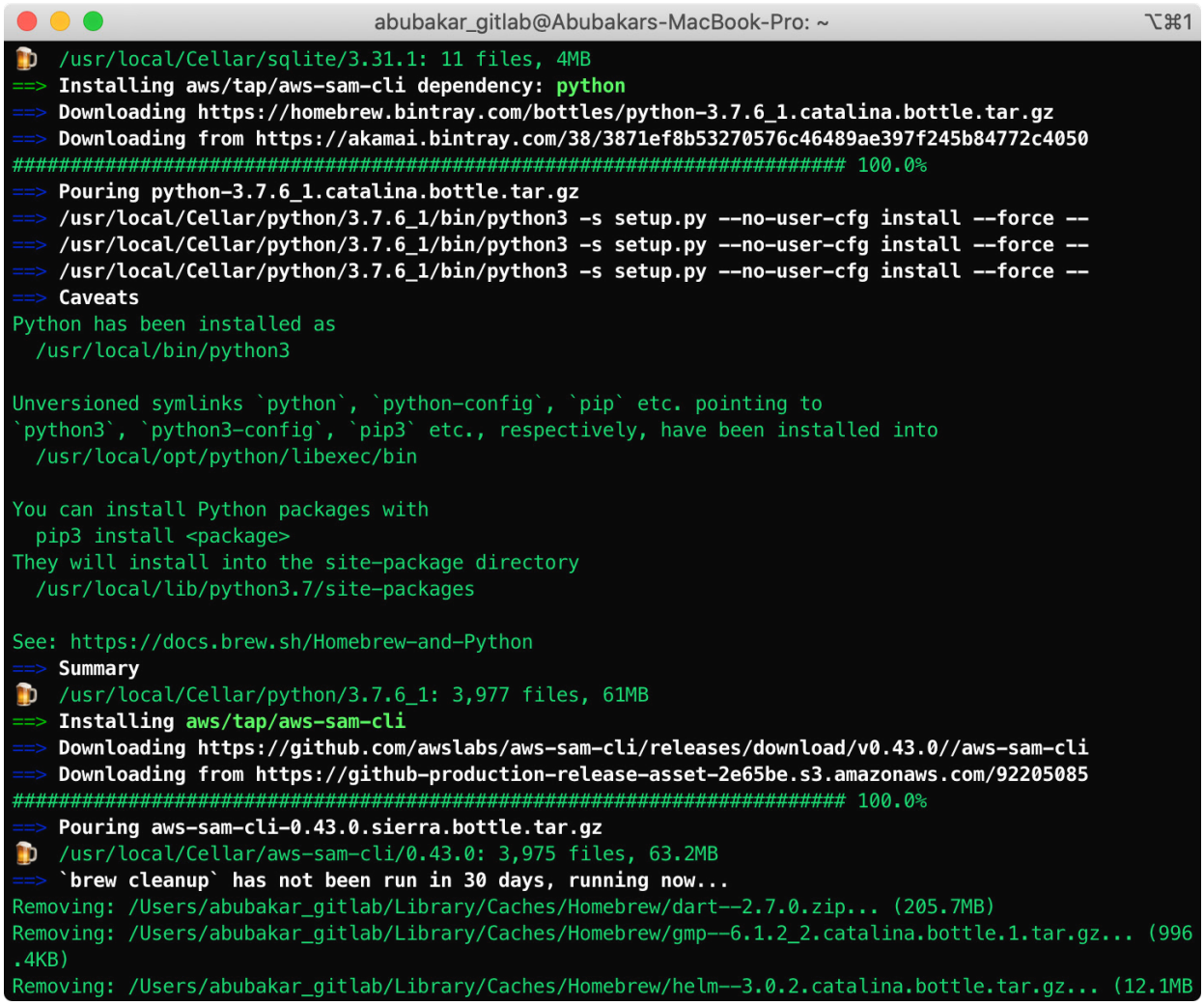
# GitLab + SAM

AWS SAM, at the highest level, is an open source framework for building serverless applications on AWS. It can be considered an extension to CloudFormation that makes it easier to define and deploy AWS resources – such as Lambda functions, API Gateway APIs and DynamoDB tables – commonly used in serverless applications.

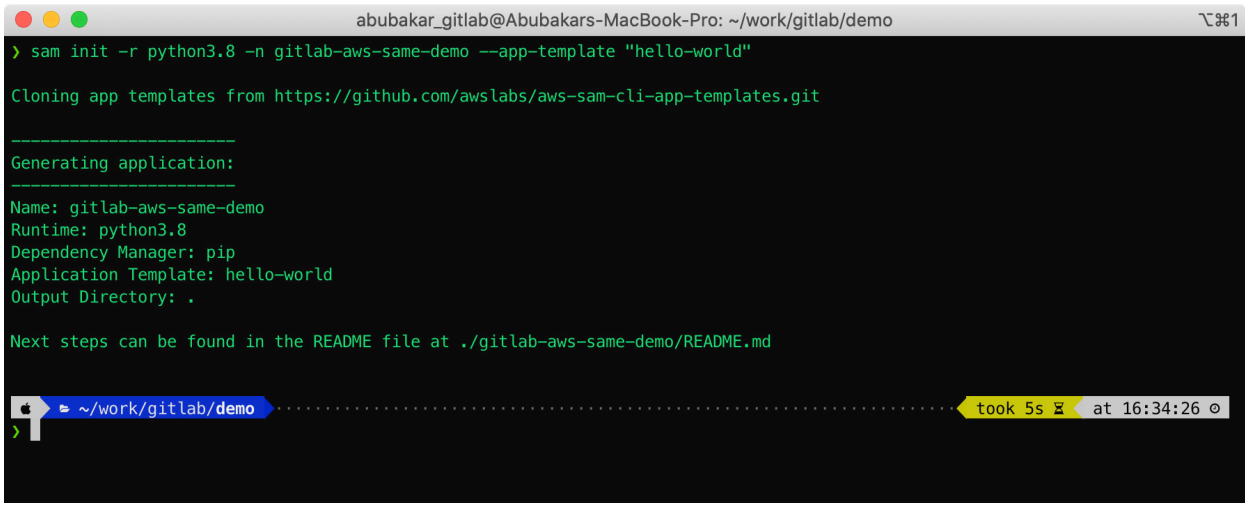
In addition to its templating capabilities, SAM also includes a CLI for testing and deployment, though some of the CLI commands are just aliases to underlying CloudFormation calls. In this sample project, we used the AWS CloudFormation CLI to build and deploy our SAM application.

We've included a sample SAM application called "AWS News" in the {{cookiecutter.project_name}} folder, but you can use the .gitlab-ci.yml file at the top of this repo with any SAM application.

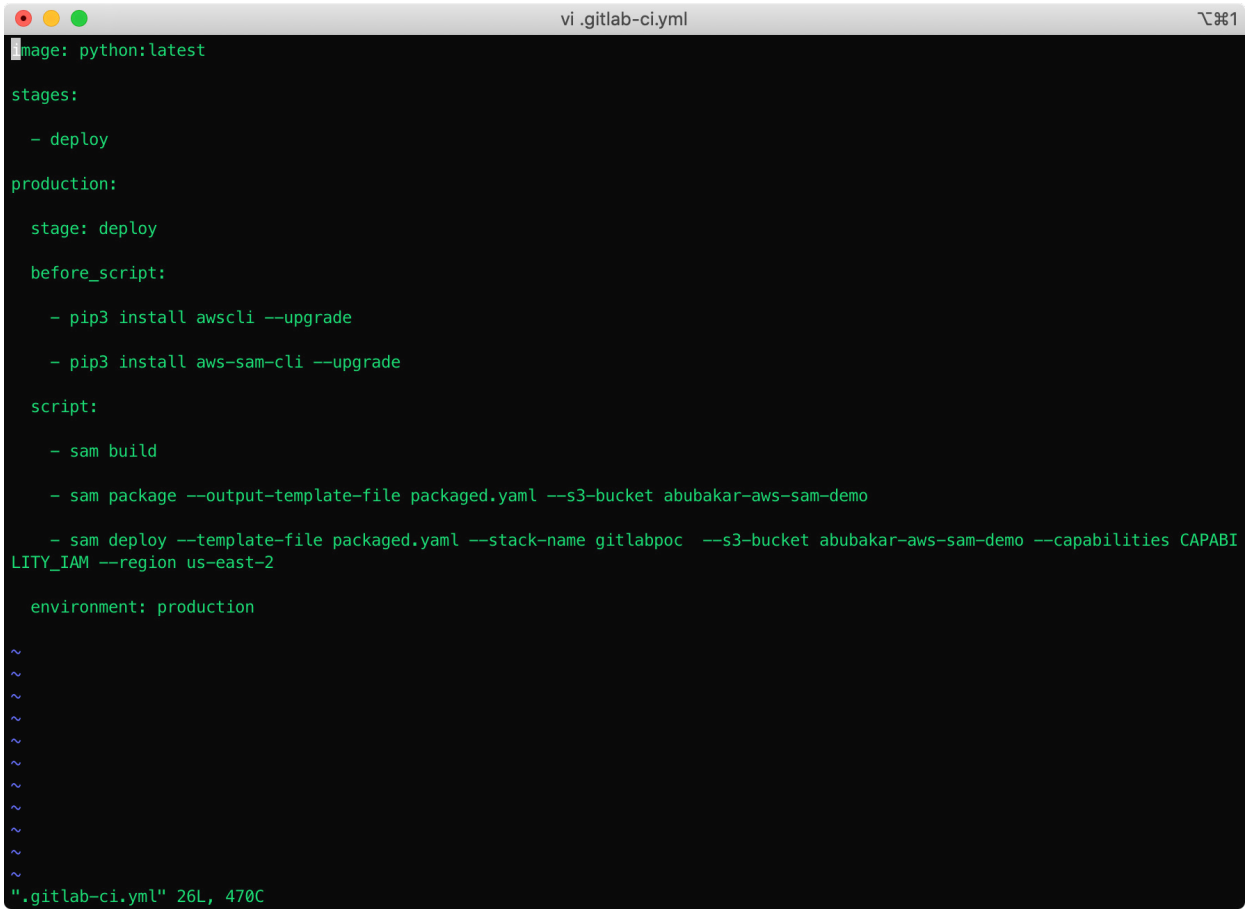
First, install AWS SAM CLI:



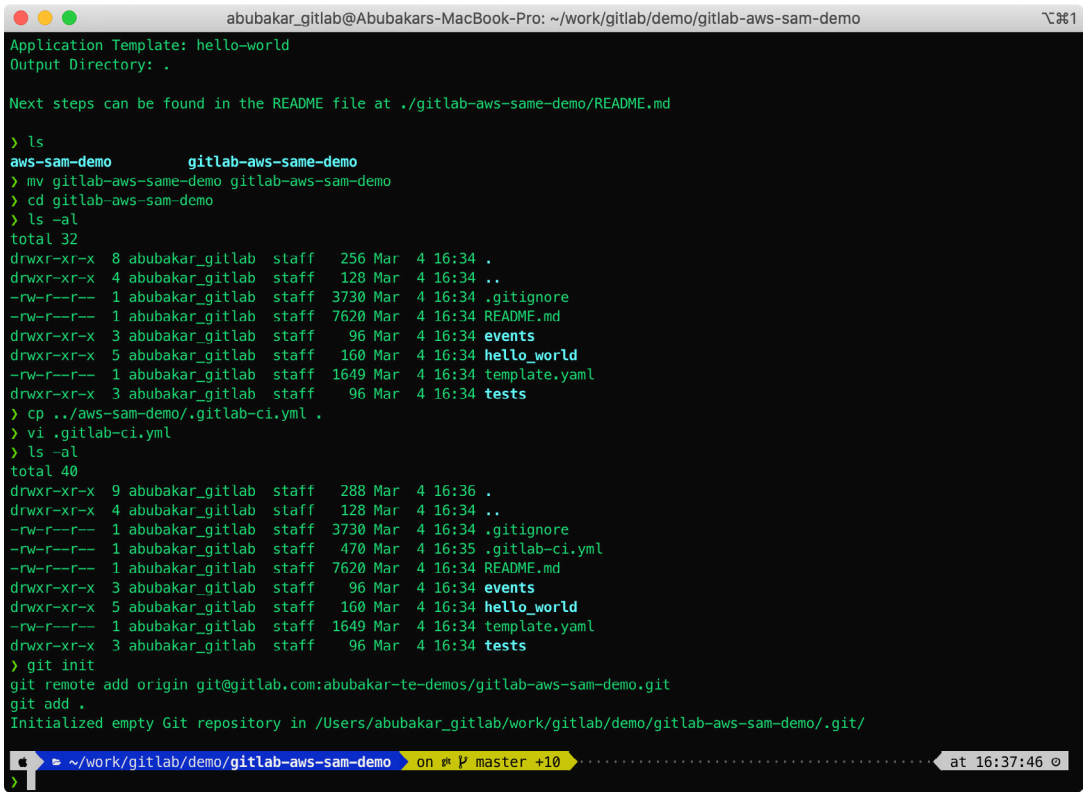
Run the SAM init command to generate AWS SAM project using a hello-world example:



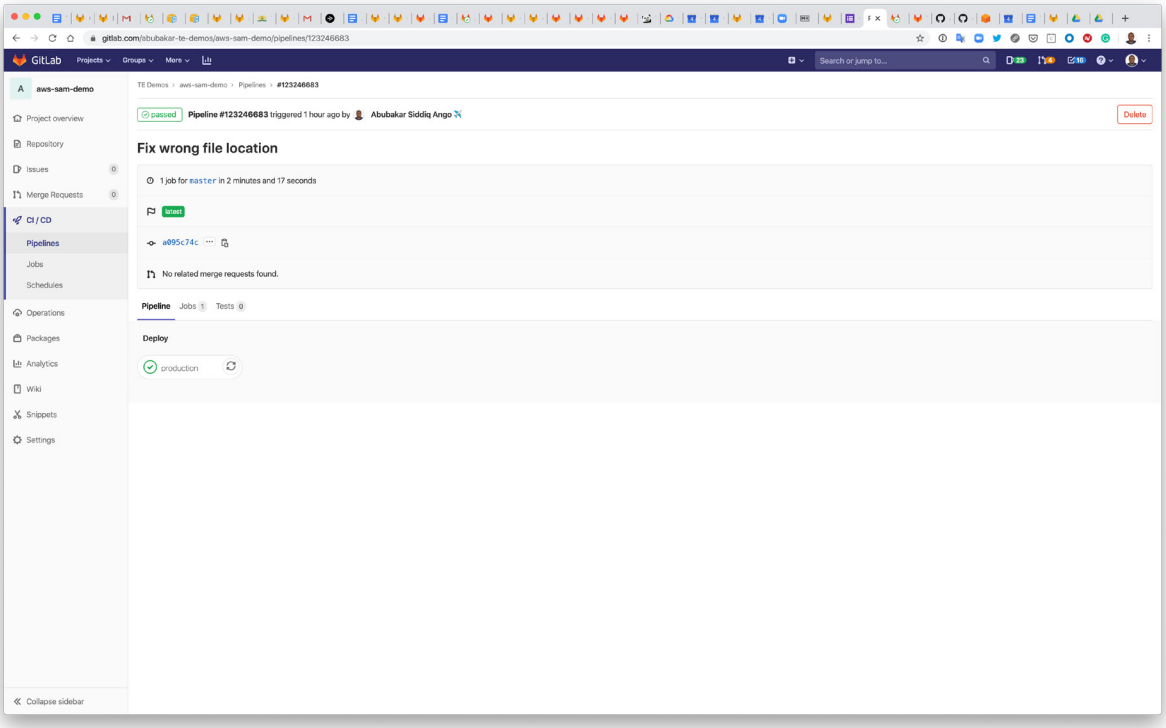

A folder is created for the project, in our casevnamed gitlab-aws-sam-demo. Next, we create a .gitlab-ci.yml file to build and deploy the project.

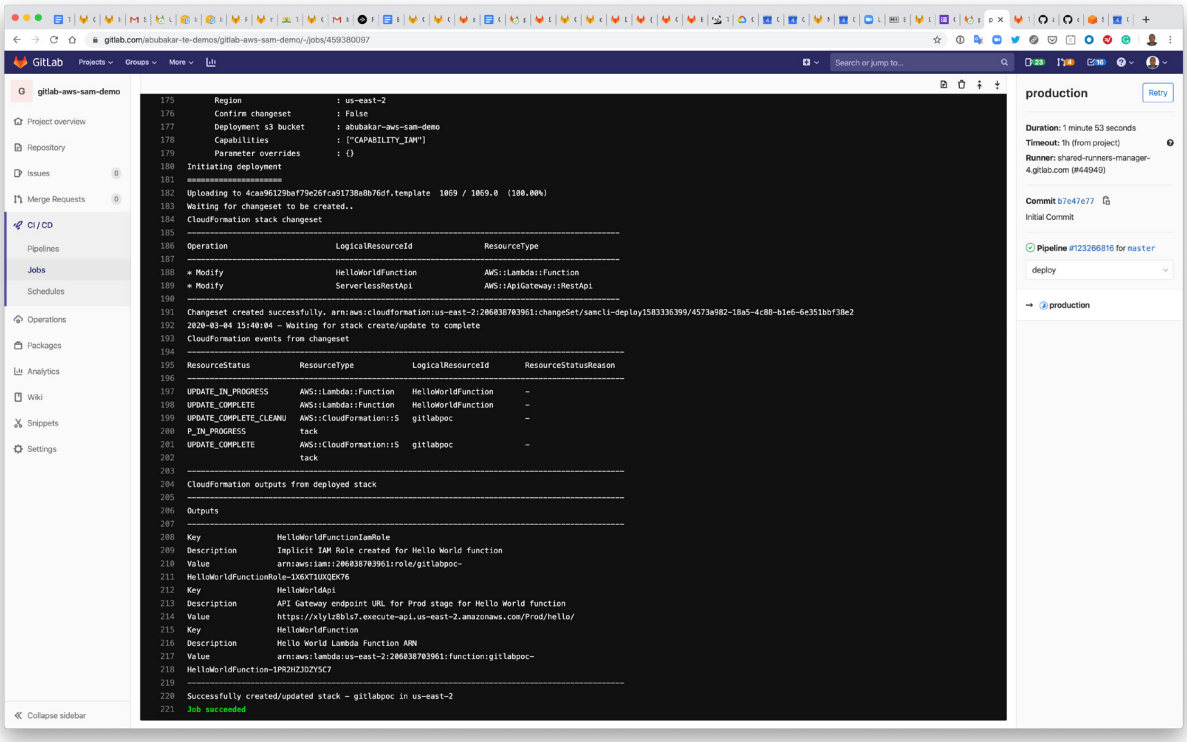


Next, we run git init and push to GitLab:




Once the pipeline is complete, it outputs the URL of the serverless endpoint as shown below:






Using curl with the Service endpoint shows the output required, which is “Hello World”.

```
abubakar_gitlab@Abubakars-MacBook-Pro: ~/work/gitlab/demo/gitlab-aws-sam-demo ⌥⌘1
-rw-r--r--  1 abubakar_gitlab  staff  7620 Mar  4 16:34 README.md
drwxr-xr-x  3 abubakar_gitlab  staff    96 Mar  4 16:34 events
drwxr-xr-x  5 abubakar_gitlab  staff   160 Mar  4 16:34 hello_world
-rw-r--r--  1 abubakar_gitlab  staff  1649 Mar  4 16:34 template.yaml
drwxr-xr-x  3 abubakar_gitlab  staff    96 Mar  4 16:34 tests
❯ git init
git remote add origin git@gitlab.com:abubakar-te-demos/gitlab-aws-sam-demo.git
git add .
Initialized empty Git repository in /Users/abubakar_gitlab/work/gitlab/demo/gitlab-aws-sam-demo/.git/
❯ git commit -m "Initial Commit"
[master (root-commit) b7e47e7] Initial Commit
 10 files changed, 606 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .gitlab-ci.yml
 create mode 100644 README.md
 create mode 100644 events/event.json
 create mode 100644 hello_world/__init__.py
 create mode 100644 hello_world/app.py
 create mode 100644 hello_world/requirements.txt
 create mode 100644 template.yaml
 create mode 100644 tests/unit/__init__.py
 create mode 100644 tests/unit/test_handler.py
❯ git push -u origin master
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 12 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (15/15), 8.16 KiB | 2.72 MiB/s, done.
Total 15 (delta 1), reused 0 (delta 0)
To gitlab.com:abubakar-te-demos/gitlab-aws-sam-demo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
❯ ls
README.md     events        hello_world   template.yaml tests
❯ curl https://xlylz8bls7.execute-api.us-east-2.amazonaws.com/Prod/hello/
{"message": "hello world"}%

 ~/work/gitlab/demo/gitlab-aws-sam-demo on  master ···· at 16:41:32
❯
```

**Additional resources:**

» serverless.yaml Configuration

» Document how to deploy serverless functions to AWS Lambda

» Deploying AWS Lambda function using GitLab CI/CD

» AWS Serverless Application Model (SAM)

» Serverless Framework

» TriggerMesh KLR as a CI/CD provider

# GitLab + Fargate

According to a recent Datadog market analysis, among companies running containers in AWS, 19% use AWS Fargate – up from 14% the year before. Fargate's rapid growth has helped Amazon Elastic Container Service keep pace with the continuing adoption of Kubernetes in AWS environments.

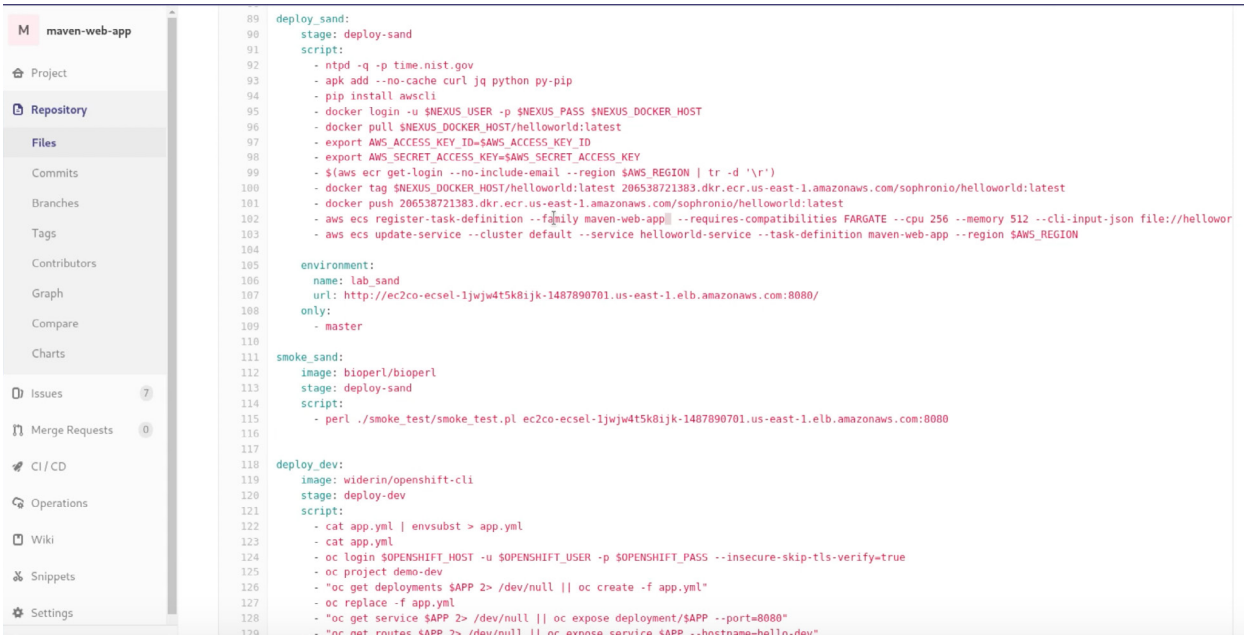
AWS Fargate is a serverless compute engine for containers that works with both Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS). Fargate primarily removes the need to provision and manage servers and allows developers to specify resources for each application.

To get started using GitLab and Fargate you'll need your AWS stack in place:

1. An Amazon ECS cluster
2. A service that belongs to the cluster that runs Fargate tasks
3. An accessible container registry for Docker images
4. An Amazon ECS task definition that references a Docker image stored in your registry that defines CPU and memory requirements
5. An application load balancer that redirects requests to healthy targets in a target group
6. A target group

Once you have these pieces in place, it's just a matter of creating a GitLab CI/CD pipeline and updating the .gitlab-ci.yml file.

Here is a sample script for a dev environment that specifies task definitions for a Fargate cluster:




**Additional resources:**

See how the team at Web Captioner was able to use GitLab and AWS for one-click Fargate deployments.

**Read their story**

# GitLab + EKS

Amazon Elastic Kubernetes Service (EKS) is a fully managed Kubernetes service. The introduction of Amazon Elastic Kubernetes Service (EKS) was widely applauded as it streamlines the abstraction of the complexities in an environment and makes creating and managing Kubernetes clusters easier with more granular controls around security and straightforward policies of how resources are used.

GitLab strives to increase developer productivity by automating repetitive tasks and allowing developers to focus on business logic. We recently introduced support for auto-creating Kubernetes clusters on Amazon EKS. GitLab also gives you the power to achieve the following use cases and more:

- Highly scalable CI/CD system using GitLab Runner: There are times like holidays when little to no changes to code are pushed to production, so why keep resources tied down? With the Amazon EKS integration with GitLab, you can install GitLab Runner with just a click and your CI/CD will run effortlessly without worrying about running out of resources.
- Shared Cluster: Maintaining multiple Kubernetes clusters can be a pain and capital intensive. With Amazon EKS, GitLab allows you to setup a cluster at Instance, Group and Project levels. Kubernetes Namespaces are created for each GitLab project when the Amazon EKS is integrated at Instance and Project level, allowing isolation and ensuring security.
- Review Apps: Reviewing changes to code or design can be tricky, you'll need to check out your branch and run the code in a test environment. GitLab integrated with Amazon EKS deploys your app with new changes to a dynamic environment and all you need to do is click on a "View App" button to review changes.
- AutoDevOps detects, builds, tests, deploys, and monitors your applications, leveraging the Amazon EKS integration. In this section, we will deploy a sample application to the Amazon EKS cluster we create using AutoDevOps.

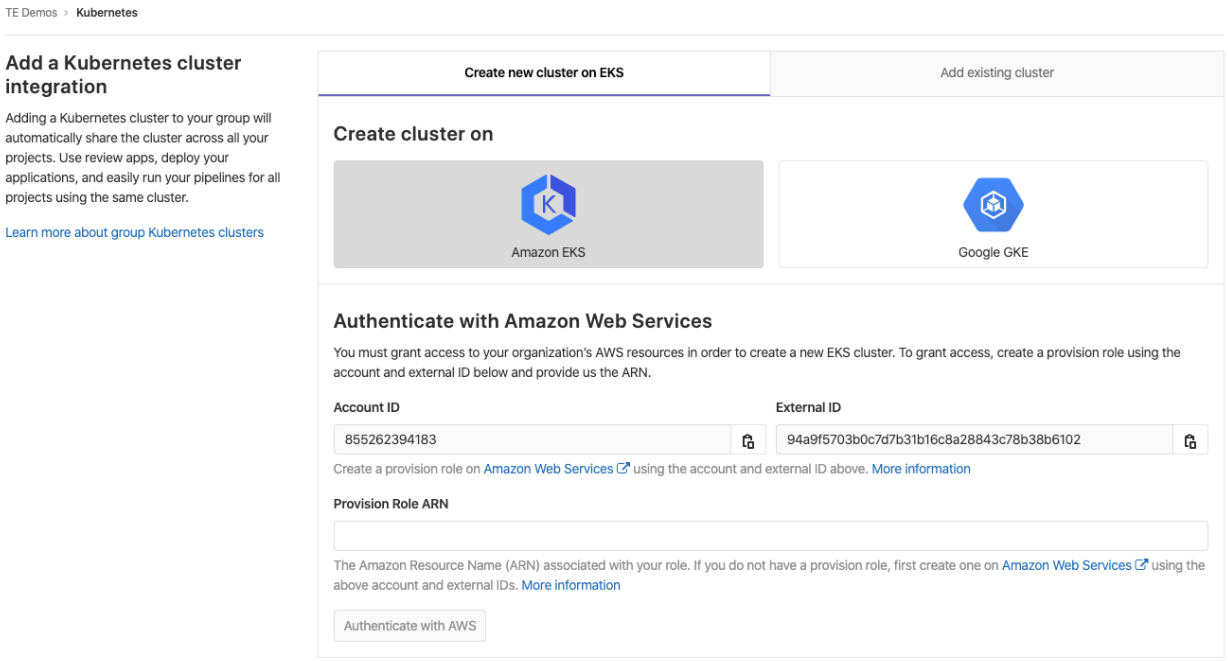
**One-time setup on AWS to access resources**

First, we need to create a "provision" role and a "service" role on AWS to grant GitLab access to AWS resources and set up the necessary permissions to create and manage EKS clusters. You only need to perform these steps once and you can reuse them anytime you want to perform another integration or create more clusters.

### Step 1 - Create provision role

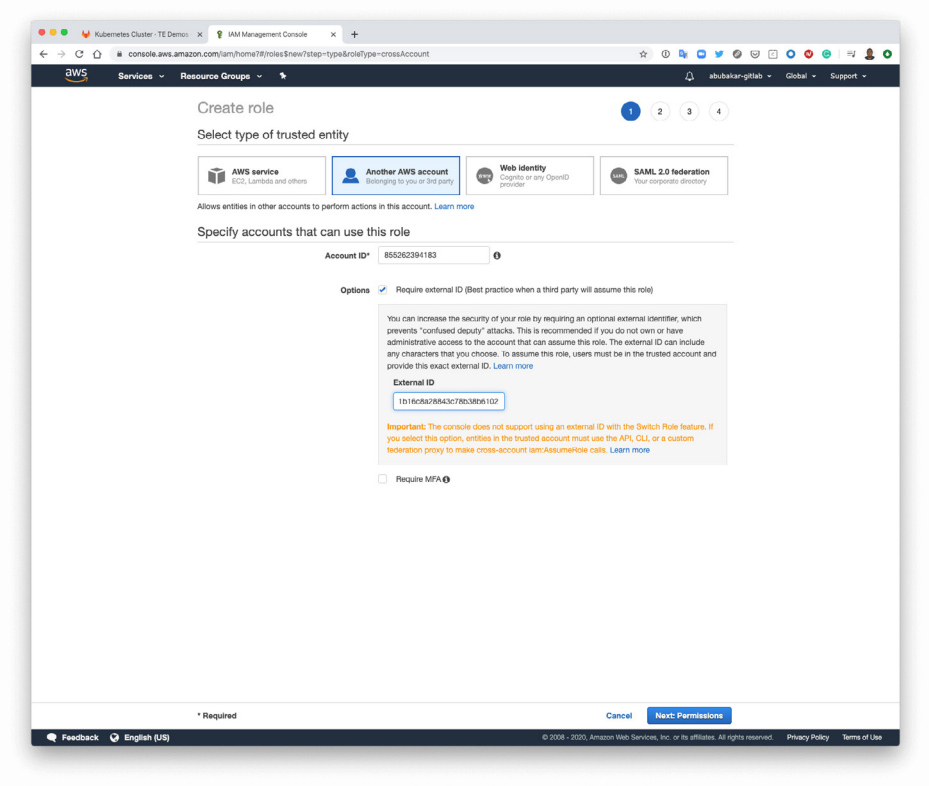
To grant GitLab access to your AWS resources, a “provision role” is required.

1. Access GitLab Kubernetes Integration Page by clicking on the ”Kubernetes” menu for groups and Operations > Kubernetes menu for projects and click the “Add Kubernetes Cluster” button.
2. Select “Amazon EKS” in the options provided under the “Create new cluster on EKS” tab.
3. You are provided with an Account and External ID to use for authentication. Make note of these values to be used in a later step.




4. Open IAM Management Console in another tab and click on “Create Role”

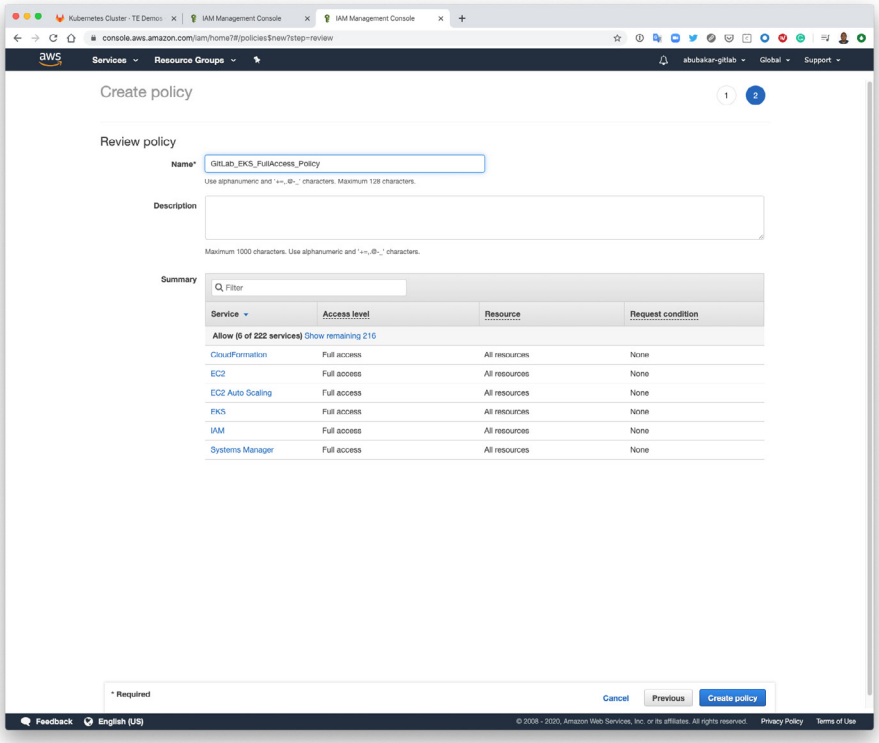
5. Click on the “Another AWS account” tab and provide the Account and External ID obtained from GitLab and click Next to set permissions as shown below:




On the permissions page, click on “Create policy.” This will open a new tab where you can set either of the permissions below using JSON:

```
{
    “Version”: “2012-10-17”,
    “Statement”: [
        {
            “Effect”: “Allow”,
            “Action”: [
                “autoscaling:*”,
                “cloudformation:*”,
                “ec2:*”,
                “eks:*”,
                “iam:*”,
                “ssm:*”
            ],
            “Resource”: “*”
        }
    ]
}
```

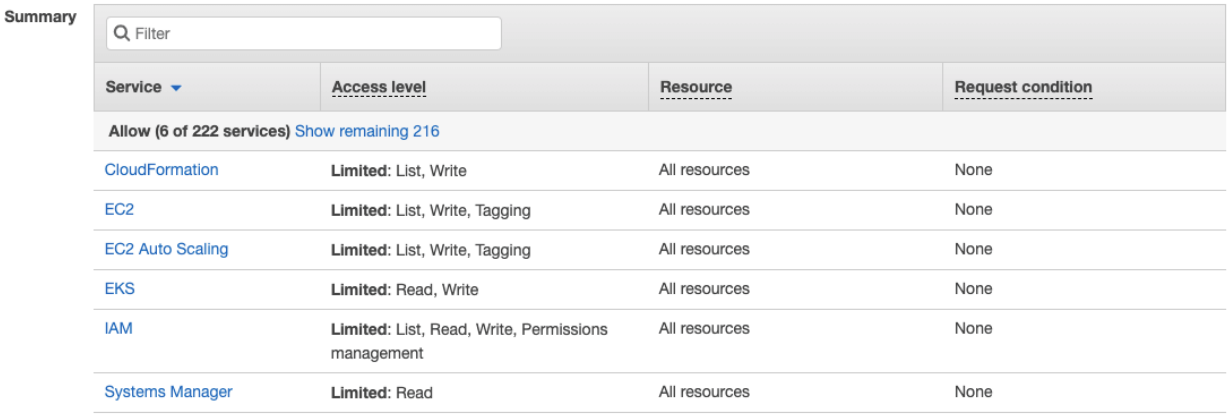
This gives GitLab full access to create and manage resources, as seen in the image below:




If you prefer limited permission, you can give GitLab the ability to create resources, but not delete them with the JSON snippet below. The drawback here is if an error is encountered during the creation process, changes will not be rolled back and you must remove resources manually.
You can do this by deleting the relevant CloudFormation stack.

```
{
    “Version”: “2012-10-17”,
    “Statement”: [
        {
            “Effect”: “Allow”,
            “Action”: [
                “autoscaling:CreateAutoScalingGroup”,
                “autoscaling:DescribeAutoScalingGroups”,
                “autoscaling:DescribeScalingActivities”,
                “autoscaling:UpdateAutoScalingGroup”,
                “autoscaling:CreateLaunchConfiguration”,
                “autoscaling:DescribeLaunchConfigurations”,
                “cloudformation:CreateStack”,
                “cloudformation:DescribeStacks”,
                “ec2:AuthorizeSecurityGroupEgress”,
                “ec2:AuthorizeSecurityGroupIngress”,
                “ec2:RevokeSecurityGroupEgress”,
                “ec2:RevokeSecurityGroupIngress”,
                “ec2:CreateSecurityGroup”,
                “ec2:createTags”,
                “ec2:DescribeImages”,
                “ec2:DescribeKeyPairs”,
                “ec2:DescribeRegions”,
                “ec2:DescribeSecurityGroups”,
                “ec2:DescribeSubnets”,
                “ec2:DescribeVpcs”,
                “eks:CreateCluster”,
                “eks:DescribeCluster”,
                “iam:AddRoleToInstanceProfile”,
                “iam:AttachRolePolicy”,
                “iam:CreateRole”,
                “iam:CreateInstanceProfile”,
                “iam:CreateServiceLinkedRole”,
                “iam:GetRole”,
                “iam:ListRoles”,
                “iam:PassRole”,
                “ssm:GetParameters”
            ],
            “Resource”: “*”
        }
    ]
}
```

6. The image below visualizes what permissions are granted:



Summary

Filter

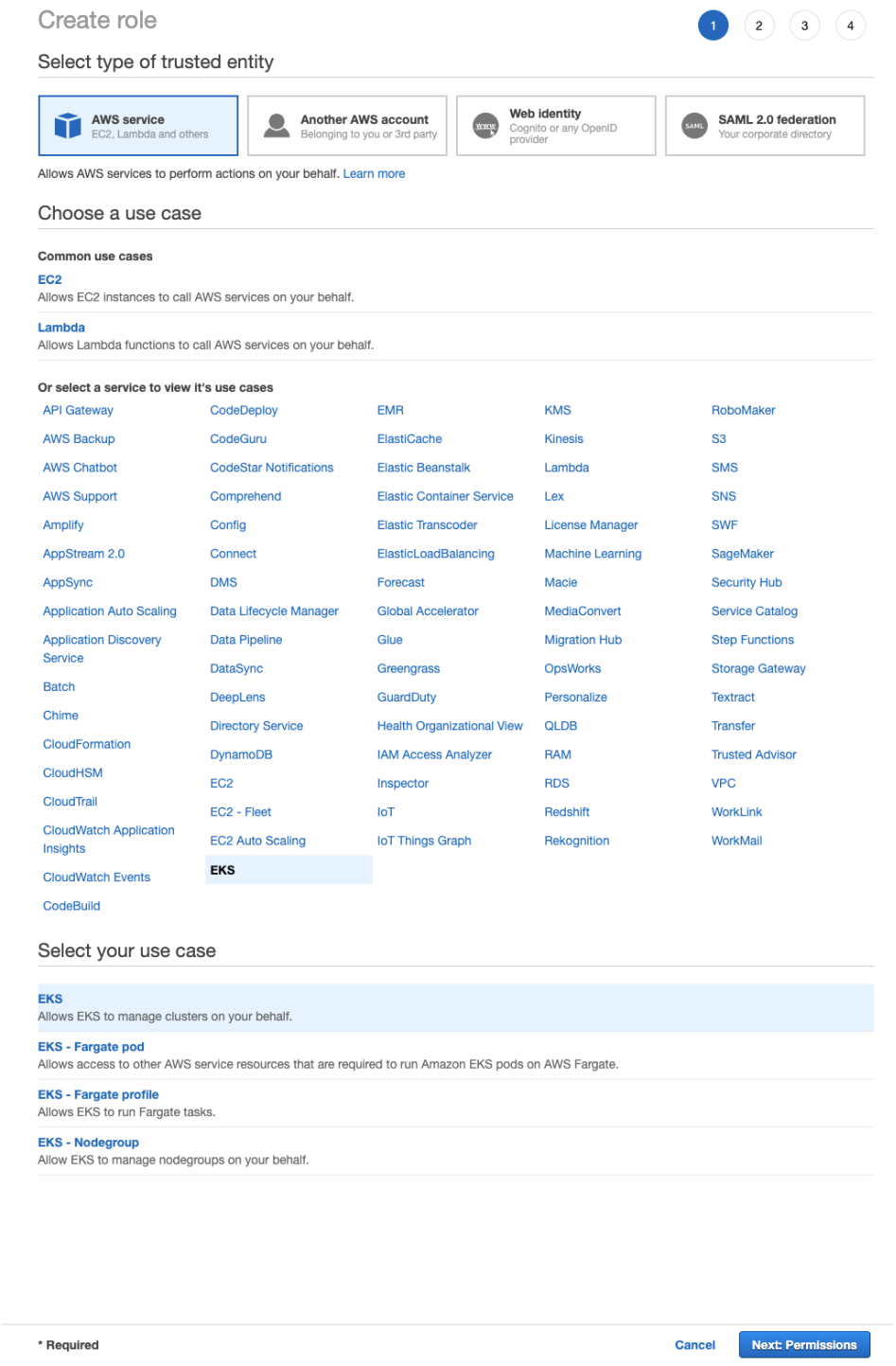
| Service | Access level | Resource | Request condition |
|---|---|---|---|
| **Allow (6 of 222 services)** Show remaining 216 | | | |
| CloudFormation | **Limited**: List, Write | All resources | None |
| EC2 | **Limited**: List, Write, Tagging | All resources | None |
| EC2 Auto Scaling | **Limited**: List, Write, Tagging | All resources | None |
| EKS | **Limited**: Read, Write | All resources | None |
| IAM | **Limited**: List, Read, Write, Permissions management | All resources | None |
| Systems Manager | **Limited**: Read | All resources | None |

7. Once the policy is created, return to the “Create Role” browser tab and refresh to see the policy we created listed. Select the policy and click “Next.”
8. In the Tags section, we don’t need to set any Tags, except if it’s required in your organization. Let’s proceed to Review.
9. Specify a Name for your new Role. You will see the policy we created listed under policies and click “Create Role” to complete the process.
10. Click on the new Role you created in the list of Roles to view its details. You may have to search for it in the list of Roles if it’s not listed in the first view. Copy the Role ARN provided – we will need it on the GitLab Kubernetes Integration page.

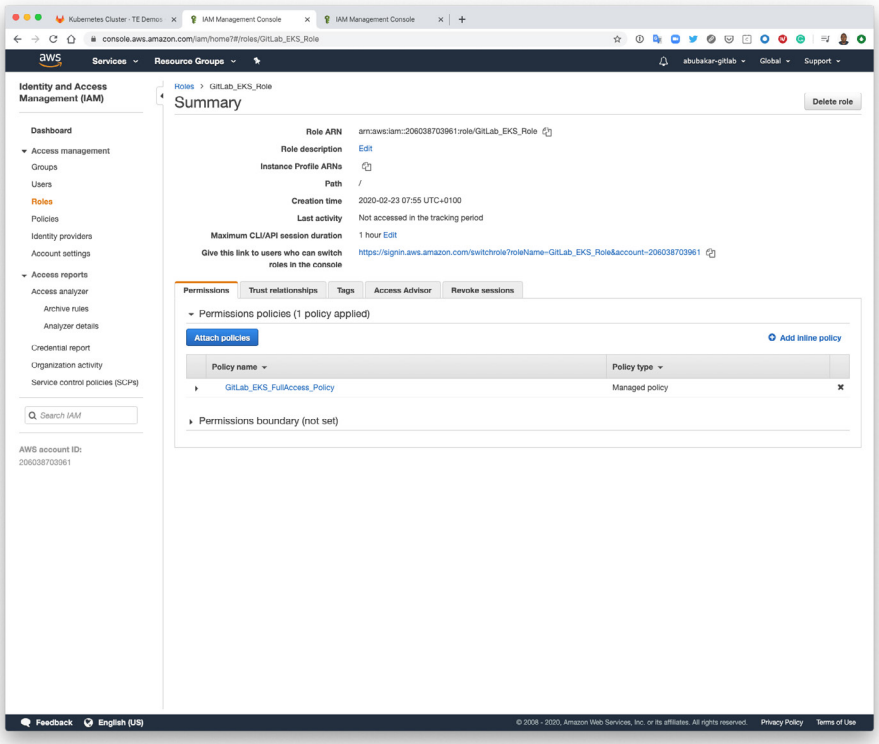
### Step 2 - Create service role

The Service Role is required to allow Amazon EKS and the Kubernetes control plane to manage AWS resources on your behalf.

1. In the IAM Management Console, click on “Create Role” and select the “AWS service” tab.
2. Select EKS in the list of services and Use Cases as shown below and click Next.



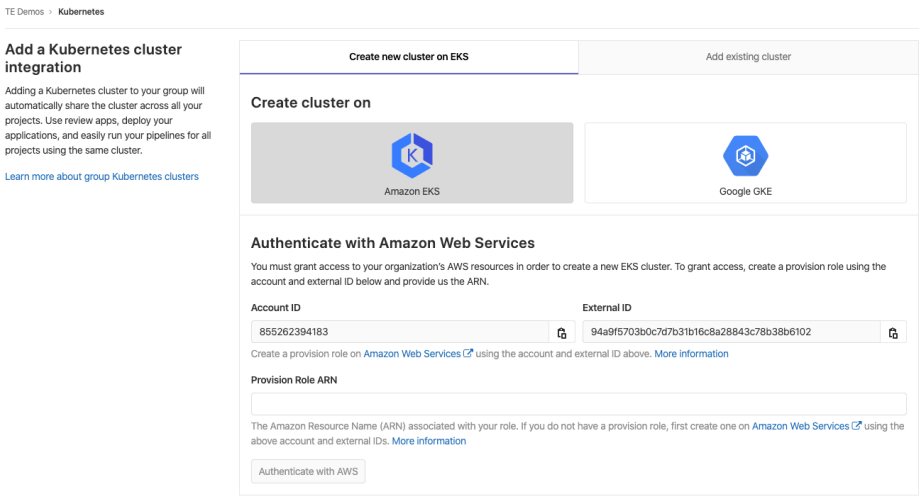
3. You will notice the “AmazonEKSClusterPolicy” and “AmazonEKSServicePolicy” permissions are selected. Click through the Tags step and create if necessary, then click Next to get to the Review step. Click “Create Role” to complete the process.



# GitLab EKS Integration

You only need to create the Provision and Service role once if you don't already have them in your organization's AWS setup. You can reuse the roles for other integrations or cluster creations.

1. Return to the GitLab Kubernetes Integration page and provide the Role ARN of the Provision Role we created earlier and click "Authenticate with AWS."




2. Once authenticated, you'll have a page to set the parameters needed to set up your cluster as shown in the image below and click on "Create Kubernetes Cluster" to let GitLab do its magic!

The parameters you'll need to provide are:

» **Kubernetes cluster name** - The name you wish to give the cluster.

» **Environment scope** - The GitLab environment associated with this cluster; * denotes the cluster will be used for deployments to all environments.

» **Kubernetes version** - The Kubernetes version to use. Currently, the only version supported is 1.14.

» **Role name** - The service role we created earlier.

» **Region** - The AWS region in which the cluster will be created.

» **Key pair name** - Select the key pair that you can use to connect to your worker nodes if required.

» **VPC** - Select a VPC to use for your EKS Cluster resources.

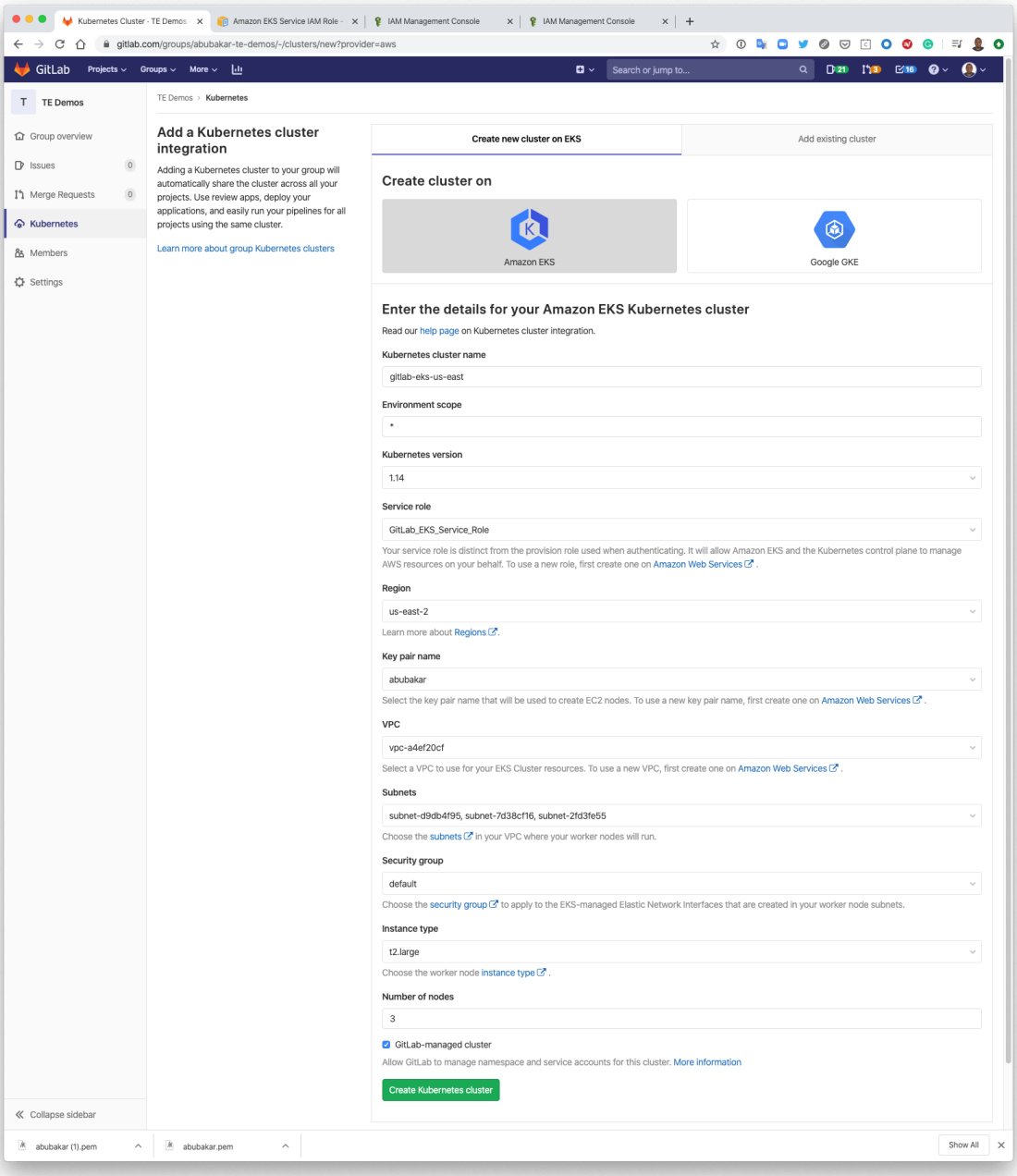
» **Subnets** - Choose the subnets in your VPC where your worker nodes will run.

» **Security group** - Choose the security group to apply to the EKS-managed Elastic Network Interfaces that are created in your worker node subnets. AWS provides a default group, which can be used for the purpose of this guide. However, you are advised to setup up the right rules required for your resources.

» **Instance type** - The AWS instance type of your worker nodes.

» **Node count** - The number of worker nodes.

» GitLab-managed cluster - Leave this checked if you want GitLab to manage namespaces and service accounts for this cluster.



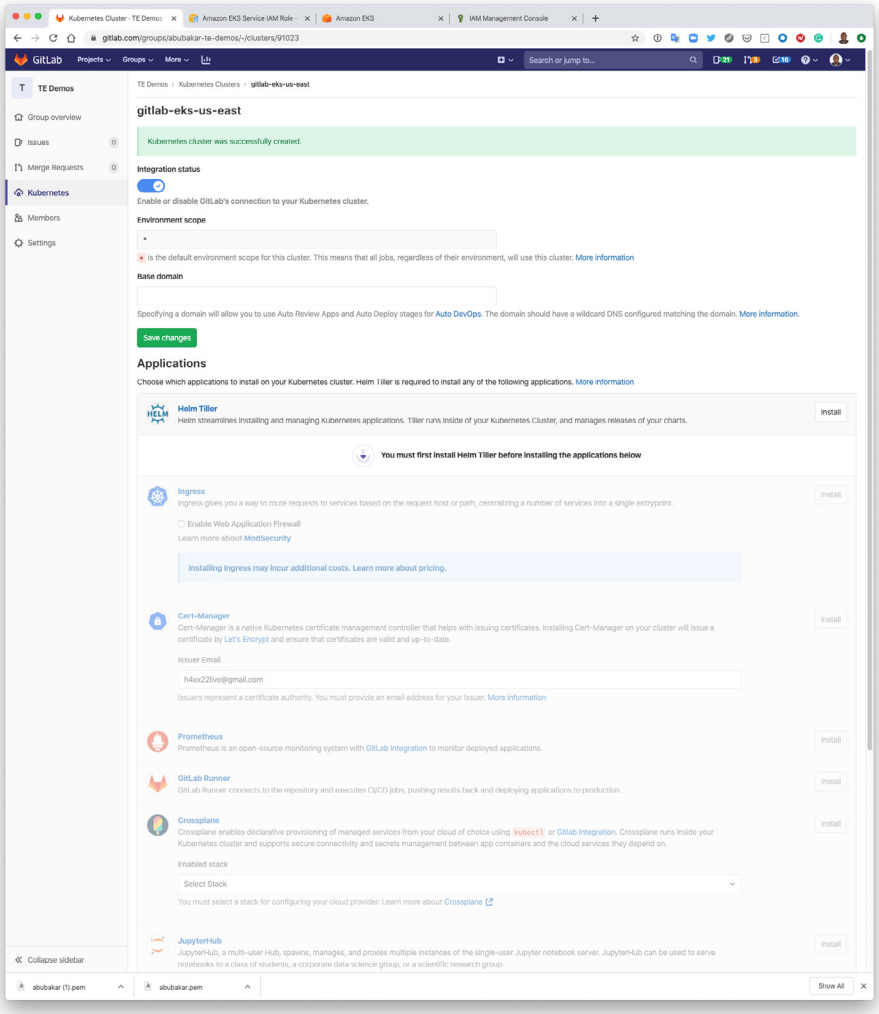
3. The cluster creation process will take approximately 10 minutes. Once done you can proceed to install some predefined applications. At the very least, you need to install the following:

» **Helm Tiller**: This is required to install the other applications.

» **Ingress**: This provides SSL termination, load balancing and name-based virtual hosting you your applications. It acts as a web proxy for your application, which is useful when using AutoDevOps or deploying your own apps.

» **Cert Manager**: This is a native Kubernetes certificate management controller, which helps in issuing certificates using Let's Encrypt. You don't need this if you want to use a custom Certificate issuer.

» **Prometheus**: GitLab uses the Prometheus integration for automatic monitoring of your applications to collect metrics from Kubernetes containers allowing you to understand what is going on from within the GitLab UI.



3. To make use of Auto Review Apps and Auto Deploy stages of AutoDevOps, you will need to specify a Base Domain name with a wild card DNS pointing to the Ingress Endpoint generated when you Install Ingress in the list of predefined apps

**Additional resources:**

» Adding and removing Kubernetes clusters

» GitLab 12.5 with EKS Cluster Creation & Environments Dashboard

» Preparing EKS resources

# GitLab + ECS

Amazon Elastic Container Service (ECS) is a fully managed container orchestration service. ECS integrates with many other services, such as AWS Identity and Access Management (IAM), and Amazon CloudWatch. ECS also gives applications the flexibility to use a mix of Amazon EC2 and AWS Fargate with Spot and On-Demand pricing options.

GitLab provides a series of CI templates that you can include in your project. To automate deployments of your application to your Amazon Elastic Container Service (ECS) cluster, you can include the Deploy-ECS.gitlab-ci.yml template in your .gitlab-ci.yml file.

Before getting started with this process, you will need a cluster on AWS ECS, as well as related components, like an ECS service, ECS task definition, a database on AWS RDS, etc.

After you're all set up on AWS ECS, follow these steps:

1. Make sure your AWS credentials are set up as environment variables for your project. You can follow the steps above to complete this setup.
2. Add these variables to your project's .gitlab-ci.yml file:

» **variables:**

» CI_AWS_ECS_CLUSTER: my-cluster

» CI_AWS_ECS_SERVICE: my-service

» CI_AWS_ECS_TASK_DEFINITION: my-task-definition

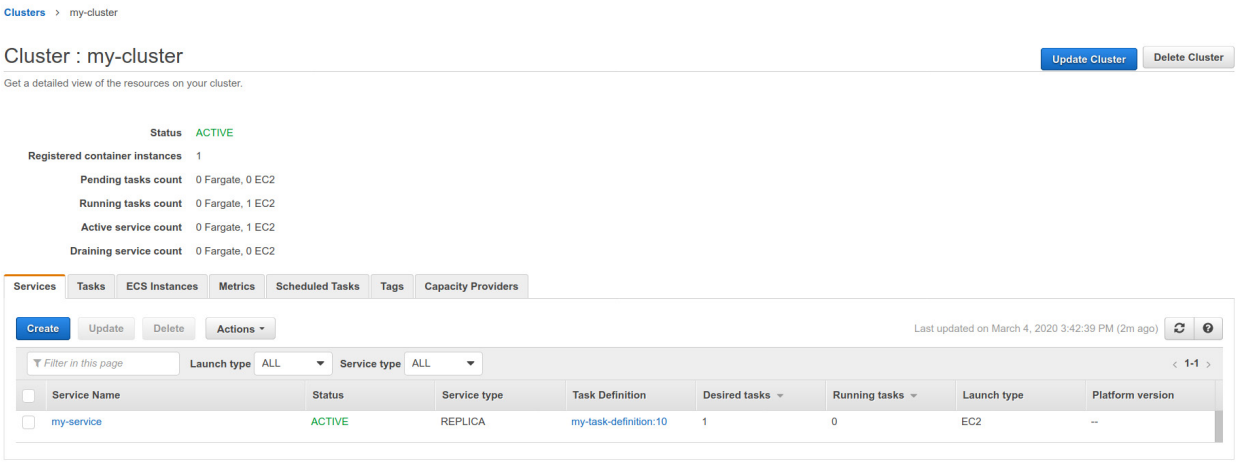
» **Three variables are defined in this snippet:**

» CI_AWS_ECS_CLUSTER: The name of your AWS ECS cluster that you're targeting for your deployments.

» CI_AWS_ECS_SERVICE: The name of the targeted service tied to your AWS ECS cluster.

» CI_AWS_ECS_TASK_DEFINITION: The name of the task definition tied to the service mentioned above.

You can find these names after selecting the targeted cluster on your AWS ECS dashboard:




3. Include this template in .gitlab-ci.yml:

```
include:
  - template: Deploy-ECS.gitlab-ci.yml
```

4. The Deploy-ECS template ships with GitLab and is available on GitLab.com.

Commit and push your updated .gitlab-ci.yml to your project's repository, and you're done!

The application Docker image will be rebuilt and pushed to the GitLab registry. The targeted task definition will be updated with the location of the new Docker image, and a new revision will be created in ECS as result.

Last, your AWS ECS service will be updated with the new revision of the task definition, making the cluster pull the newest version of your application.

**Additional resources:**

» Deploy your application to AWS Elastic Container Service (ECS)

» Automatically deploy to ECS (Amazon Container Service) (#39089) · Issues · GitLab.org

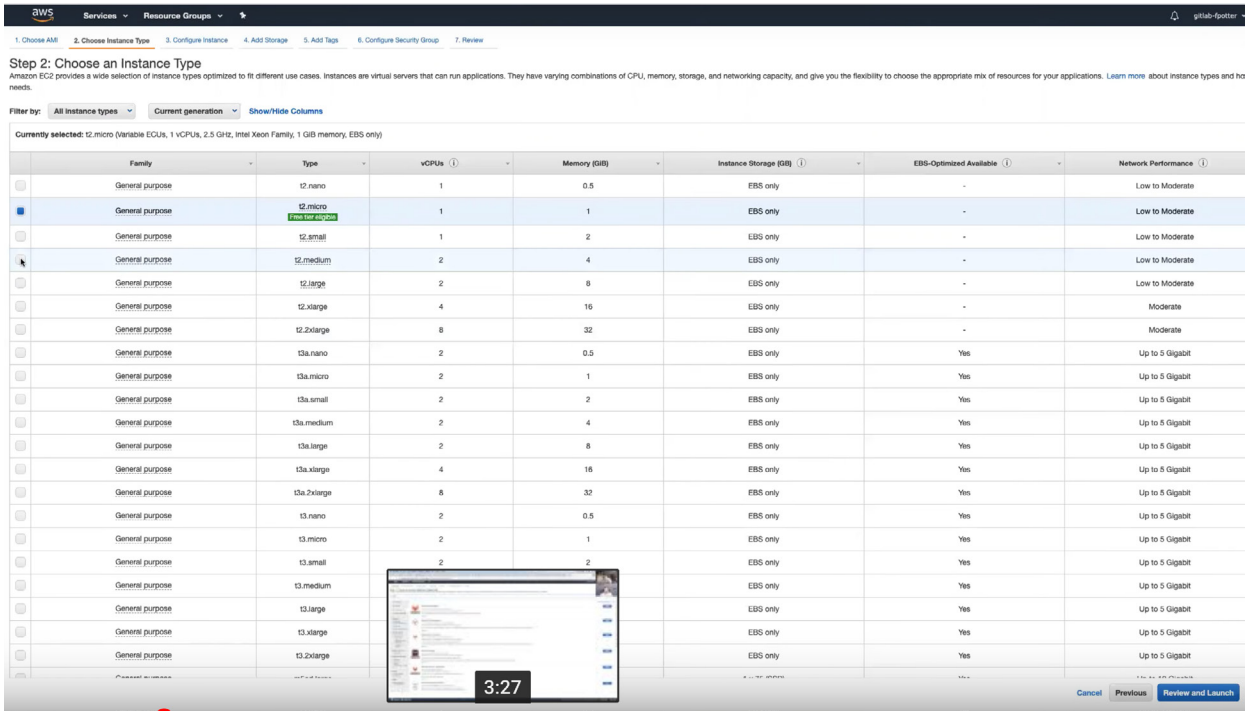
» How to configure a Gitlab / ECS continuous deployment pipeline

# GitLab + EC2

Amazon Elastic Compute Cloud (EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

If you would like to connect EC2 and GitLab from the AWS Console, select an Amazon Machine Image (AMI) from the available templates.

Choose an instance type optimized to fit your unique use cases:




Your GitLab instance is going to need to talk to the Runners over the network, and that is something you need to think about when configuring any AWS security groups or when setting up your DNS configuration.

Developers can also use Terraform to spin up an AWS instance to easily target EC2. If you would like more information, we've included a sample Terraform project to launch on AWS EC2.

GitLab can be a useful tool for utilizing EC2 spot instances. Amazon EC2 Spot instances allow you to bid on spare Amazon EC2 computing capacity. Since Spot instances are often available at a discount compared to On-Demand pricing, you can significantly reduce the cost of running your applications, grow your application's compute capacity and throughput for the same budget, and enable new types of cloud computing applications.

Substrakt Health, a technology company that provides digital solutions for the National Health Service (NHS), used Autoscaling GitLab Runners to save 90% on their EC2 costs.

**Read their story**

**Additional resources:**

» Creating an IAM EC2 instance role and profile

» Deploy Code to AWS EC2 Instances Automatically with GitLab

» Autoscaling GitLab Runner on AWS EC2

# Additional AWS functionality in the works

» 12.10 Autoscaling GitLab CI jobs on AWS Fargate

» 13.0 Template for Deploying to AWS EC2

» 13.1 Offer in-product guidance for deploying to AWS

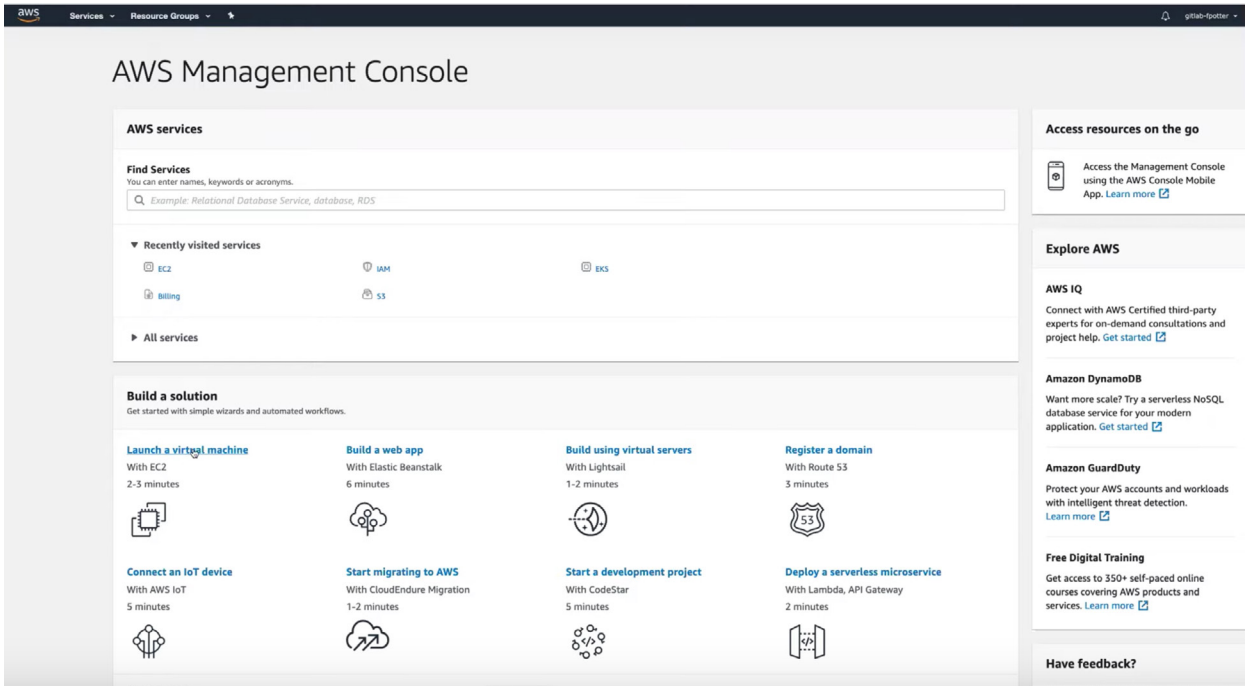
GitLab releases new features on the 22nd of every month. For a list of release posts including patch releases, please check the blog category releases. Future releases, and their important features, can be found on our upcoming releases page. You can also view upcoming features by product tier.

# Getting started with GitLab

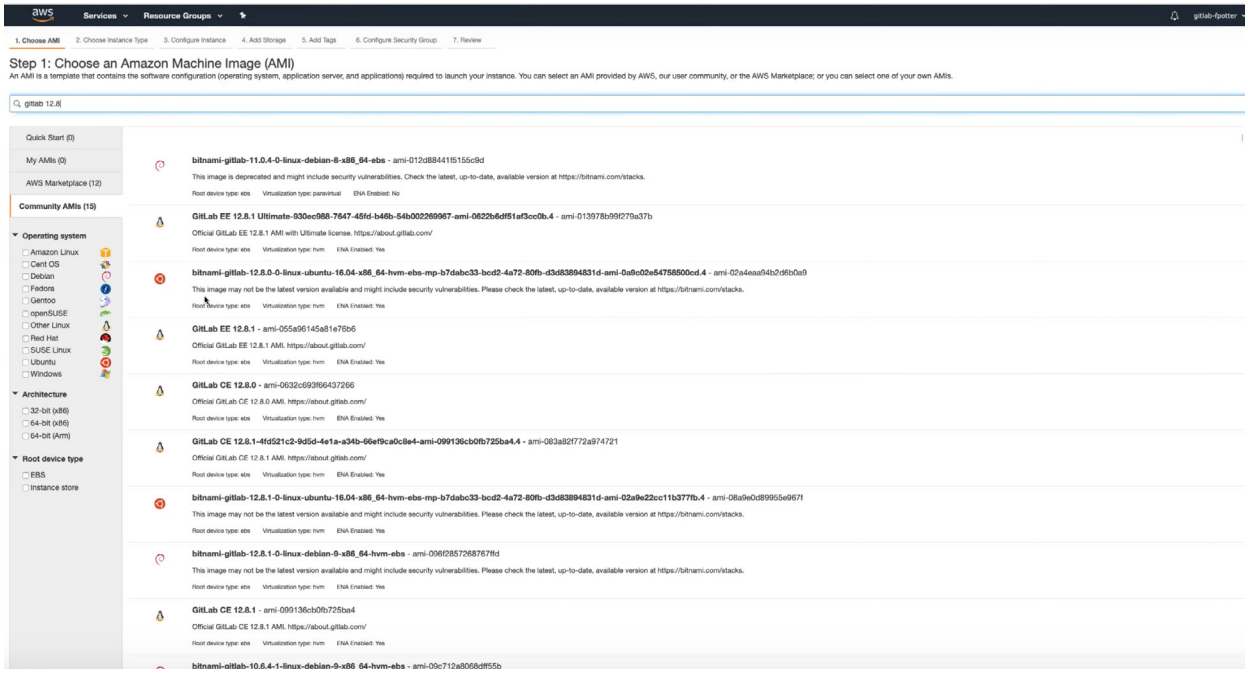
Changing DevOps processes can be an organizational challenge, so a proof of concept is essential for evaluating feasibility. Every team has different capabilities and priorities, and new processes should be properly vetted. Luckily, teams looking to try GitLab for their AWS deployments have a few options.

# Installing GitLab from AWS Management console

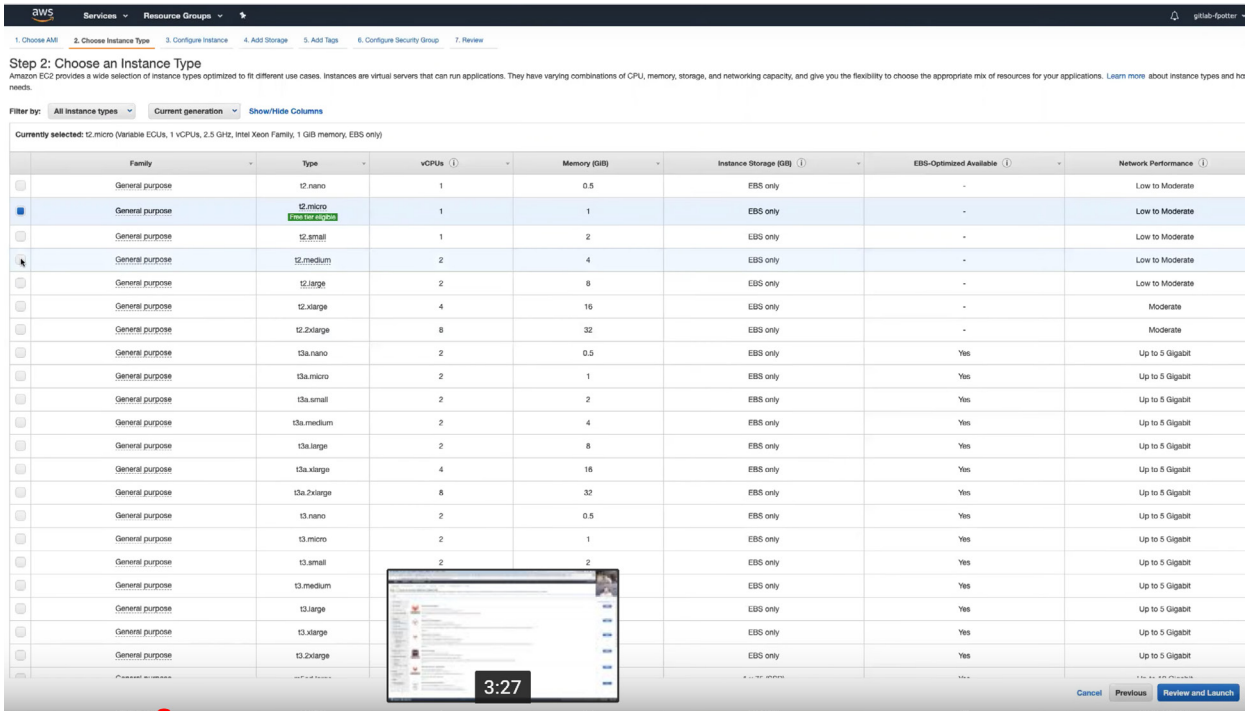
Go the AWS Management Console and select **LAUNCH A VIRTUAL MACHINE**



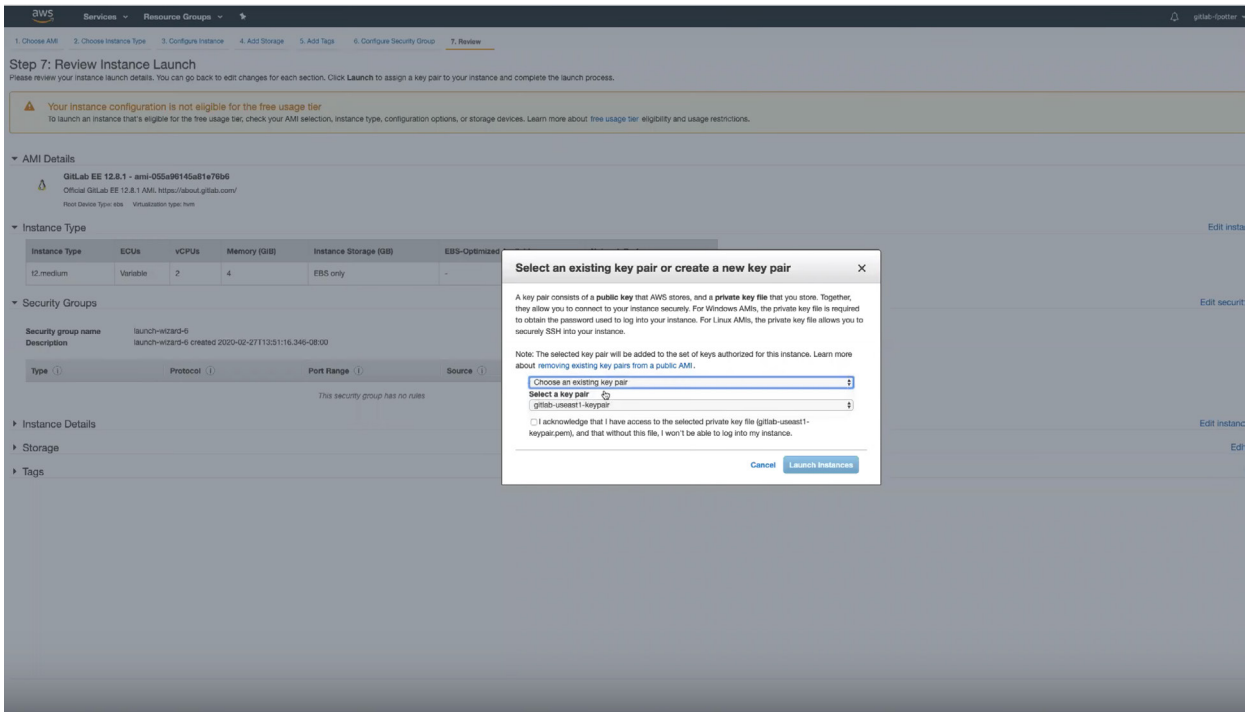

Next, click on **COMMUNITY AMIS** and search for the latest version of GitLab (ex: GitLab 12.8, GitLab 12.9 etc). Click **SELECT**



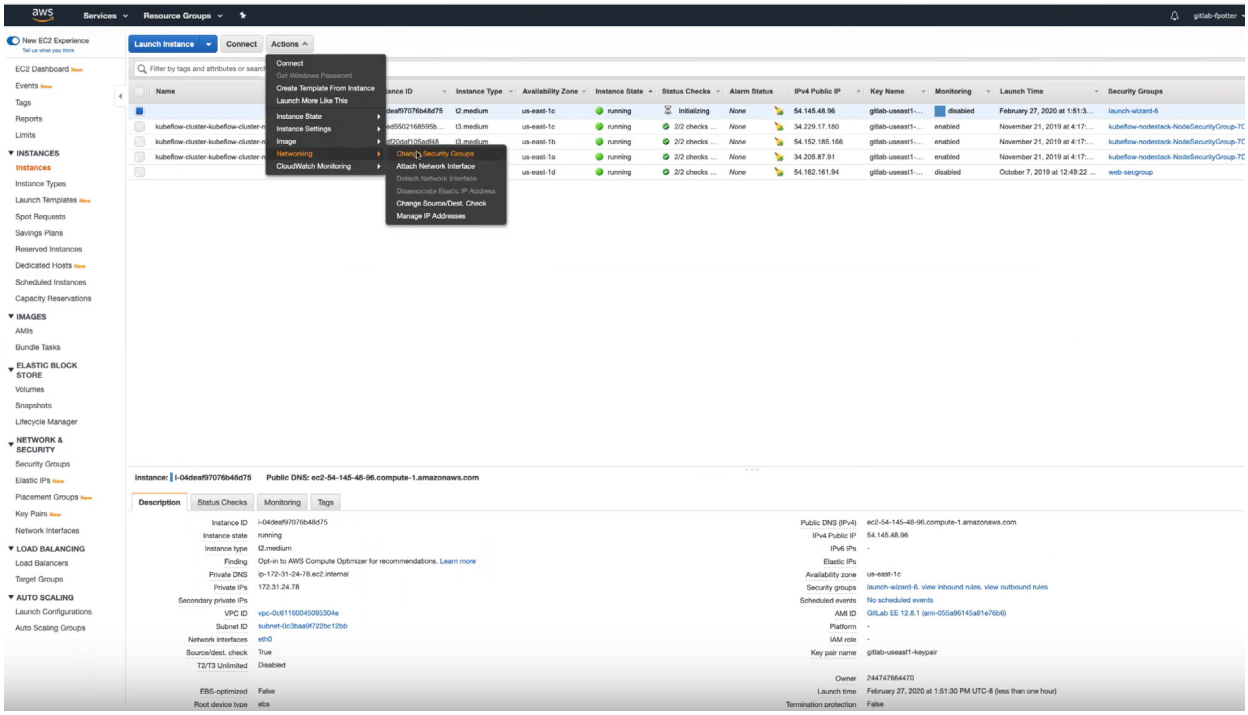
At the **CHOOSE AN INSTANCE TYPE**, select your instance. GitLab recommends **T2.MEDIUM** or higher




Select an existing key pair or create a new key pair and select **LAUNCH INSTANCES**

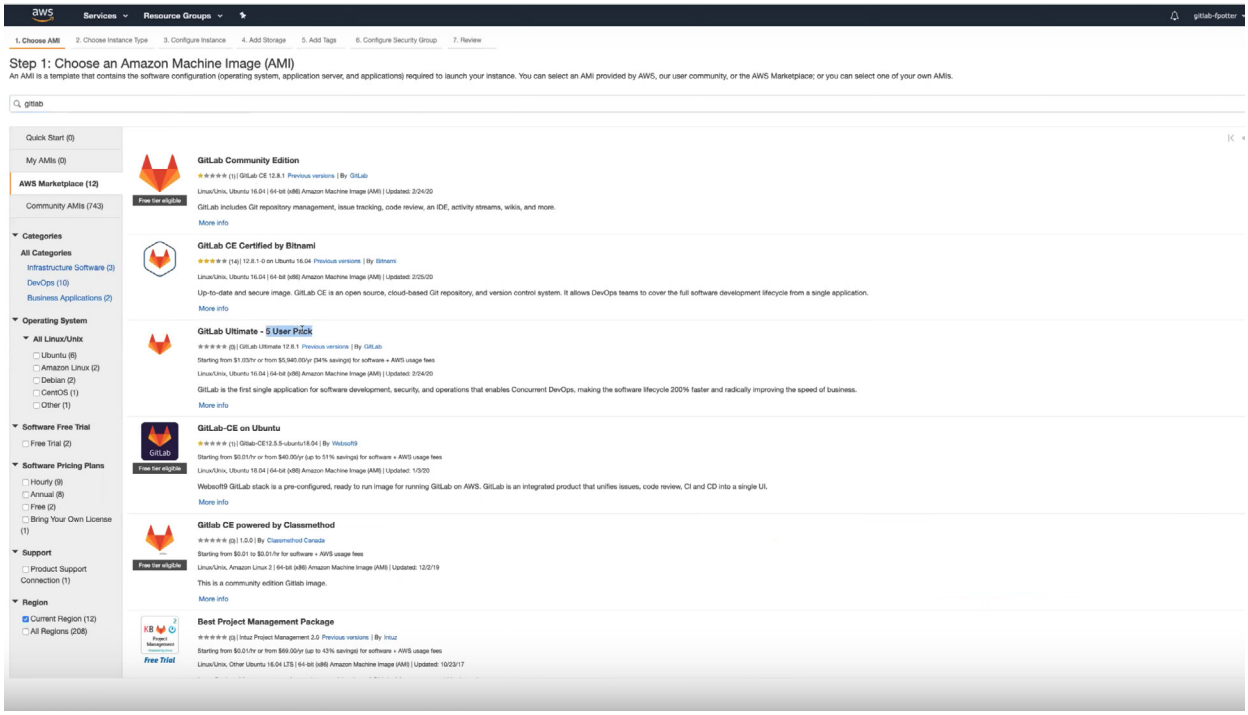


You're done! You should now be able to access your GitLab Community Edition instance from the browser. If it does not appear, make sure to check the security group.



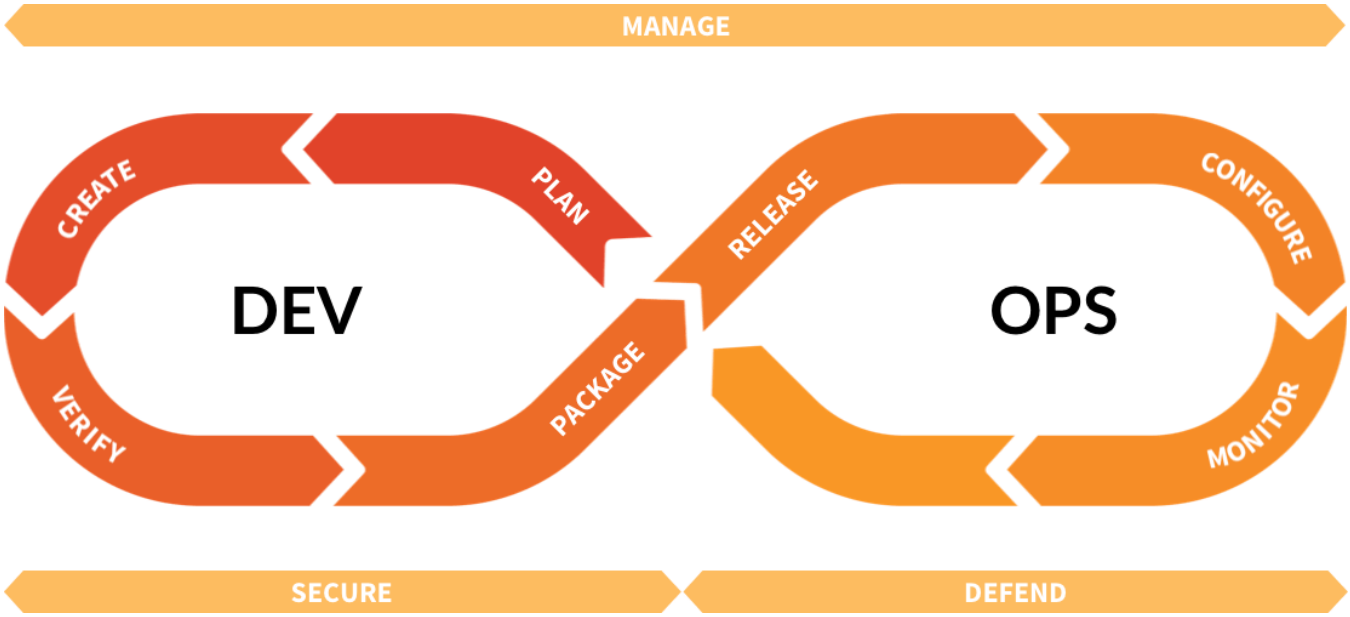
# Installing GitLab from AWS Marketplace

Another option is going through the AWS Marketplace. When you search for GitLab, select the **GITLAB ULTIMATE - 5 USER PACK** option. Finish the setup prompts and you're done.




Note: the GitLab Ultimate - 5 User Pack is not a free trial on the AWS Marketplace, but you will receive a GitLab license, as well as access to our Ultimate features. This is a great option if you would like to try the full GitLab experience with a limited number of users first.

# Benefits of using GitLab for AWS services




Organizations love AWS because it serves as a one-stop-shop for all cloud computing and IT needs. An all-in-one cloud platform can improve efficiency by bringing visibility into cloud processes and cloud usage. DevOps tools can make a big impact on cloud usage, but If development is bogged down by brittle toolchains and inefficient DevOps processes, teams won't be able to manage resources effectively.

GitLab accelerates software delivery by driving higher efficiency across all stages of the software development lifecycle. Increased visibility means teams are more productive, reducing handoffs and eliminating bottlenecks. Development, Security, and Ops teams can collaborate in one interface and deploy to any AWS infrastructure.

**All-in-one cloud meets all-in-one DevOps**

By running GitLab on AWS you get a complete DevOps platform running and deploying to AWS cloud infrastructure. GitLab and AWS offer similar benefits to organizations looking to streamline their development process.

- **Better visibility**: Teams using AWS can utilize multiple cloud services on a single platform, and DevOps teams can build, stage and deploy to the services from one application using GitLab.
- **Increased efficiency**: All-in-one cloud means teams don't have to manage multiple cloud platforms, and GitLab's single interface means teams can collaborate across all stages of the SDLC.
- **Reduced cycle times**: Instead of managing multiple clouds, processes, and tools, teams can utilize AWS and GitLab and focus on building software (not maintenance).

As organizations continue their journey around digital transformation, DevOps has become the go-to set of best practices to increase their velocity of delivering value to the business while increasing quality. Breaking down the walls and reducing the handoffs necessary to complete a full delivery lifecycle is also very important and remains a challenge to most organizations.

GitLab is a complete DevOps platform, delivered as a single application with bring-your-own-infrastructure flexibility. By running GitLab on AWS you get a complete DevOps platform, delivered as a single application, running and deploying to your AWS cloud infrastructure.

Sign up for a 30-day free trial

# About GitLab

GitLab is a DevOps platform built from the ground up as a single application for all stages of the DevOps lifecycle enabling Product, Development, QA, Security, and Operations teams to work concurrently on the same project.

GitLab provides teams a single data store, one user interface, and one permission model across the DevOps lifecycle allowing teams to collaborate and work on a project from a single conversation, significantly reducing cycle time and focus exclusively on building great software quickly.

Built on Open Source, GitLab leverages the community contributions of thousands of developers and millions of users to continuously deliver new DevOps innovations. More than 100,000 organizations from startups to global enterprise organizations, including Ticketmaster, Jaguar Land Rover, NASDAQ, Dish Network and Comcast trust GitLab to deliver great software at new speeds. GitLab is the world's largest all-remote company, with more than 1,200 team members in over 65 countries.


GitLab