

How to deploy on AWS from GitLab

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



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

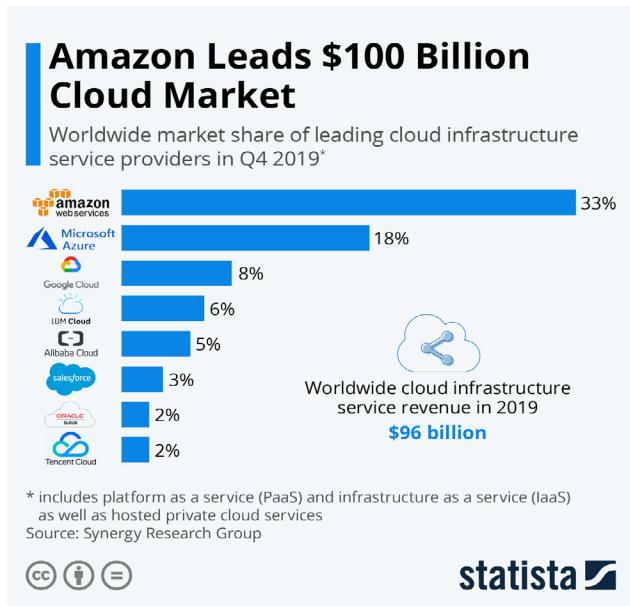
Getting started with GitLab

- » 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.

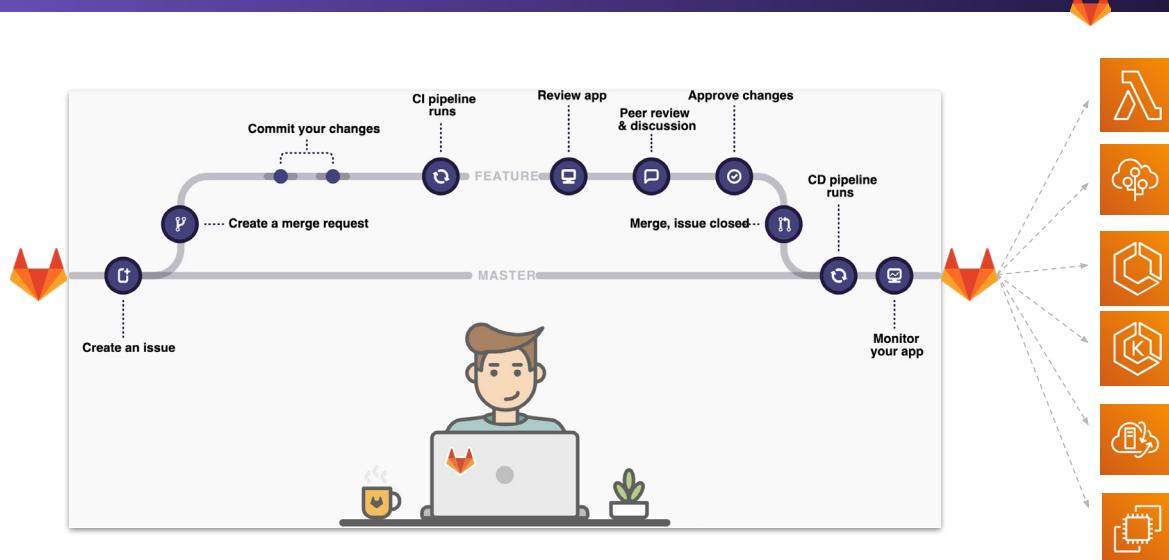
Teams that go “all-in” on AWS do so in order to simplify their cloud needs, but because AWS offers so many cloud services, this simplicity does not come at the cost of functionality.

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



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

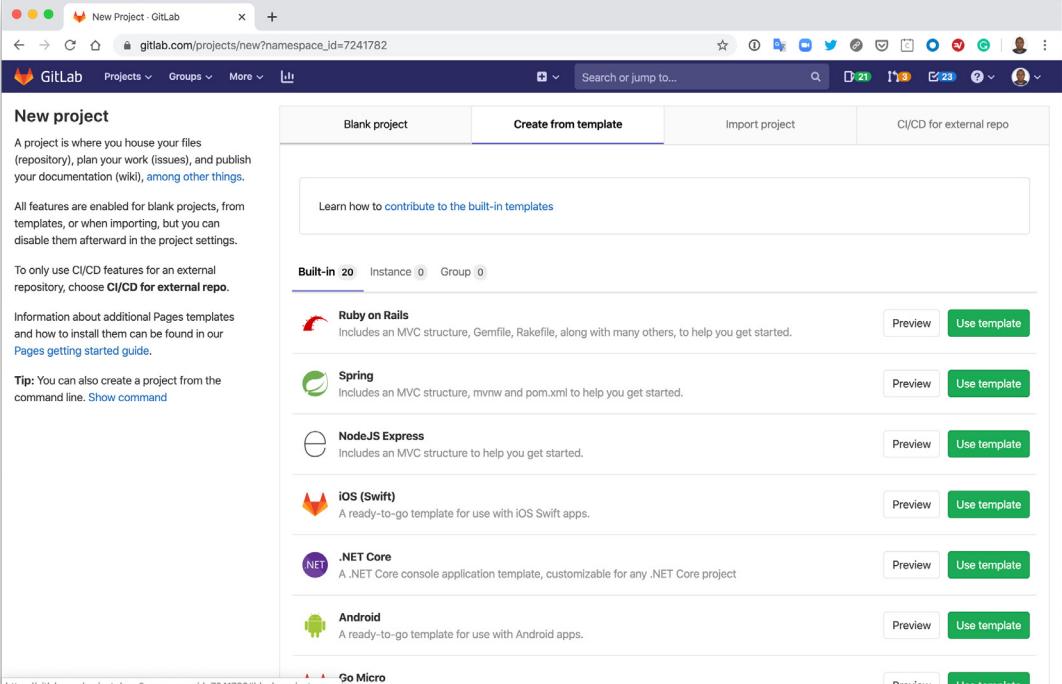
Lambda is an AWS proprietary compute service that integrates with other AWS services as well as open source solutions. The majority of developers building serverless applications today rely on Lambda functions.

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.

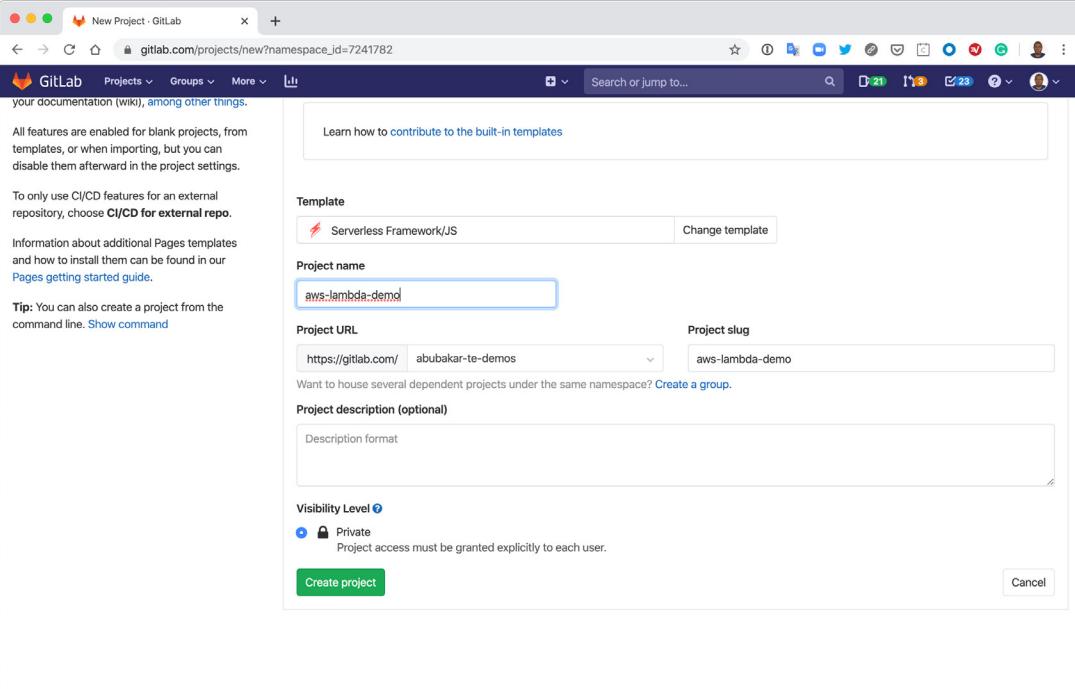
GitLab allows users to deploy AWS Lambda functions and create rich serverless applications using a combination of:

- » 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:



The screenshot shows the 'Create from template' section of the GitLab 'New project' interface. At the top, there are four tabs: 'Blank project', 'Create from template' (which is selected), 'Import project', and 'CI/CD for external repo'. Below the tabs, a box contains the text: 'Learn how to contribute to the built-in templates'. Under the heading 'Built-in 20', there are ten template cards, each with a preview button and a 'Use template' button. The templates listed are: Ruby on Rails, Spring, NodeJS Express, iOS (Swift), .NET Core, and Android. At the bottom of the list is a 'Go Micro' template.



The screenshot shows the 'Create project' dialog box. The 'Template' field is set to 'Serverless Framework/JS'. The 'Project name' field contains 'aws-lambda-demo'. The 'Project URL' field shows 'https://gitlab.com/ abubakar-te-demos' and the 'Project slug' field shows 'aws-lambda-demo'. A note below the project URL says 'Want to house several dependent projects under the same namespace? Create a group.' The 'Project description (optional)' field is empty. Under 'Visibility Level', the 'Private' radio button is selected with the note 'Project access must be granted explicitly to each user.' At the bottom right of the dialog are 'Create project' and 'Cancel' buttons.

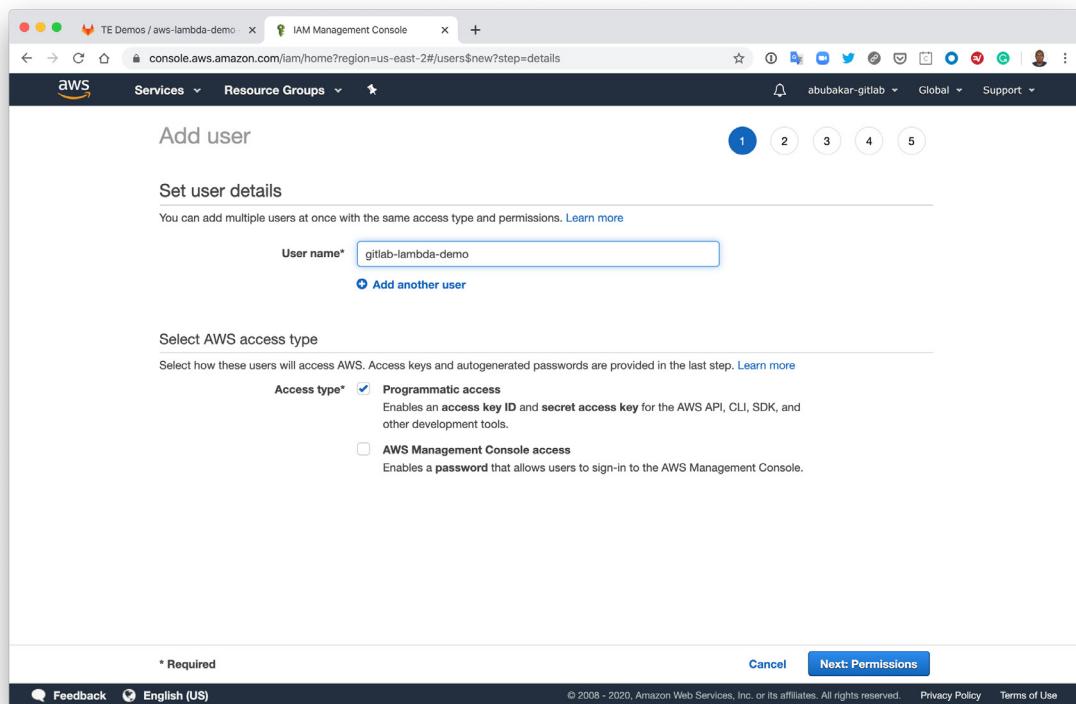
The screenshot shows the GitLab interface for creating a new project. The top navigation bar includes 'Projects', 'Groups', 'More', and a search bar. Below the header, a sidebar lists various template categories: 'Pages/Hexo', 'Netlify/Hugo', 'Netlify/Jekyll', 'Netlify/Plain HTML', 'Netlify/GitBook', 'Netlify/Hexo', 'SalesforceDX', and 'Serverless Framework/JS'. Each template entry features a preview button and a 'Use template' button.

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

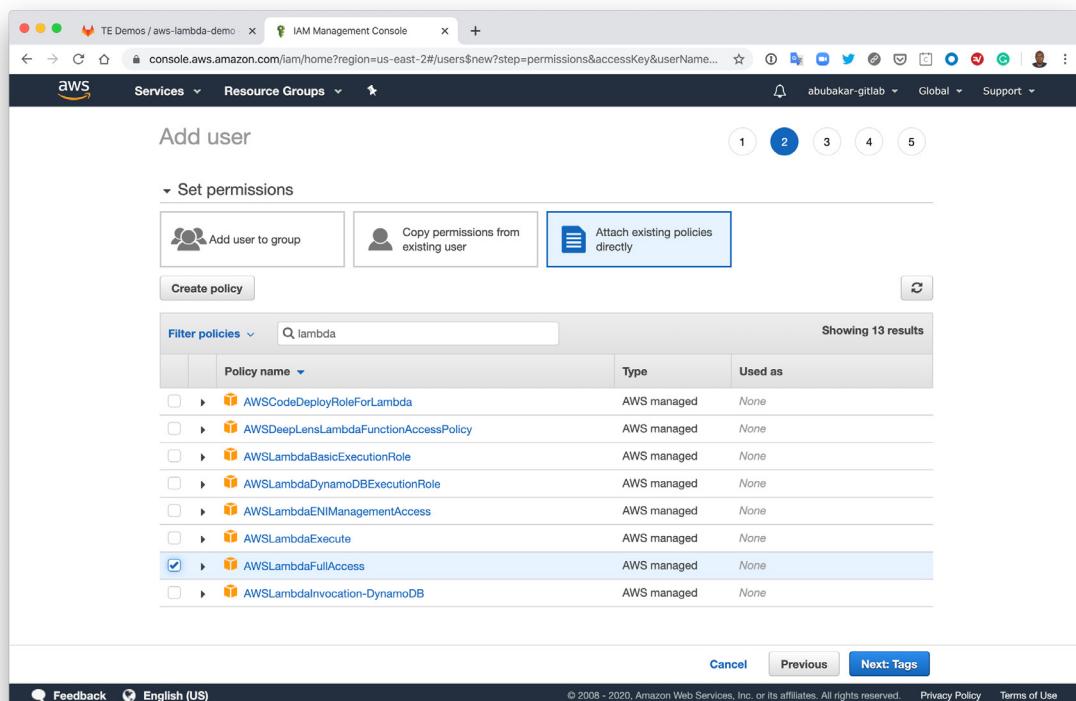
The screenshot shows the 'Details' page for the 'aws-lambda-demo' project. The sidebar on the left contains links for 'Project overview', 'Repository', 'Issues', 'Merge Requests', 'CI / CD', 'Operations', 'Packages', 'Wiki', 'Snippets', and 'Settings'. The main content area displays a message: 'The project was successfully imported.' Below this, the project name 'aws-lambda-demo' is shown with a lock icon, indicating it's a protected project. It has a Project ID of 17237218 and was authored by GitLab 3 months ago. The repository stats show 1 Commit, 1 Branch, 0 Tags, and 205 KB Files. A commit history table is present, showing the following files and their last update times:

Name	Last commit	Last update
featureTests	Initialized from 'Serverless Framework/JS' project template	3 months ago
public	Initialized from 'Serverless Framework/JS' project template	3 months ago
src	Initialized from 'Serverless Framework/JS' project template	3 months ago
.gitignore	Initialized from 'Serverless Framework/JS' project template	3 months ago
.gitlab-ci.yml	Initialized from 'Serverless Framework/JS' project template	3 months ago

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



Attach the necessary policies to the user:



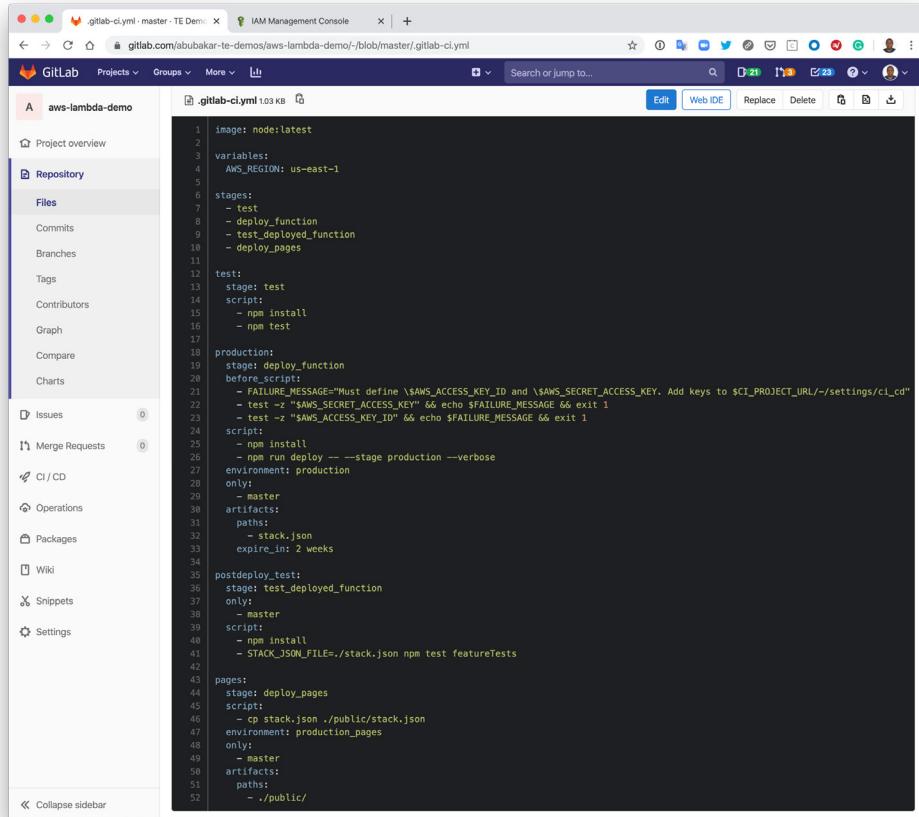
Once these steps are completed, you should have the following summary:

The screenshot shows the AWS IAM Management Console with a success message: "Success: You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time." Below the message, there is a table with one row containing a user named "gitlab-lambda-demo". The table has columns for User, Access key ID, and Secret access key. The Access key ID is listed as "AKIAS76G5D5M2RQ3CKMU" with a "Show" link next to it. A "Download .csv" button is located above the table.

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

The screenshot shows the GitLab CI/CD Settings page for the project "aws-lambda-demo". Under the "Variables" section, there are three variables defined: "AWS_ACCESS_KEY_" (Value: masked, State: Protected, Masked: yes, Scope: All environments), "AWS_SECRET_ACCE" (Value: masked, State: Protected, Masked: yes, Scope: All environments), and "Input variable key" (Value: masked, State: Protected, Masked: yes, Scope: All environments). There are "Save variables" and "Reveal values" buttons at the bottom of the table. Below the table, there is a "Group variables (inherited)" section and a "Pipeline triggers" section.

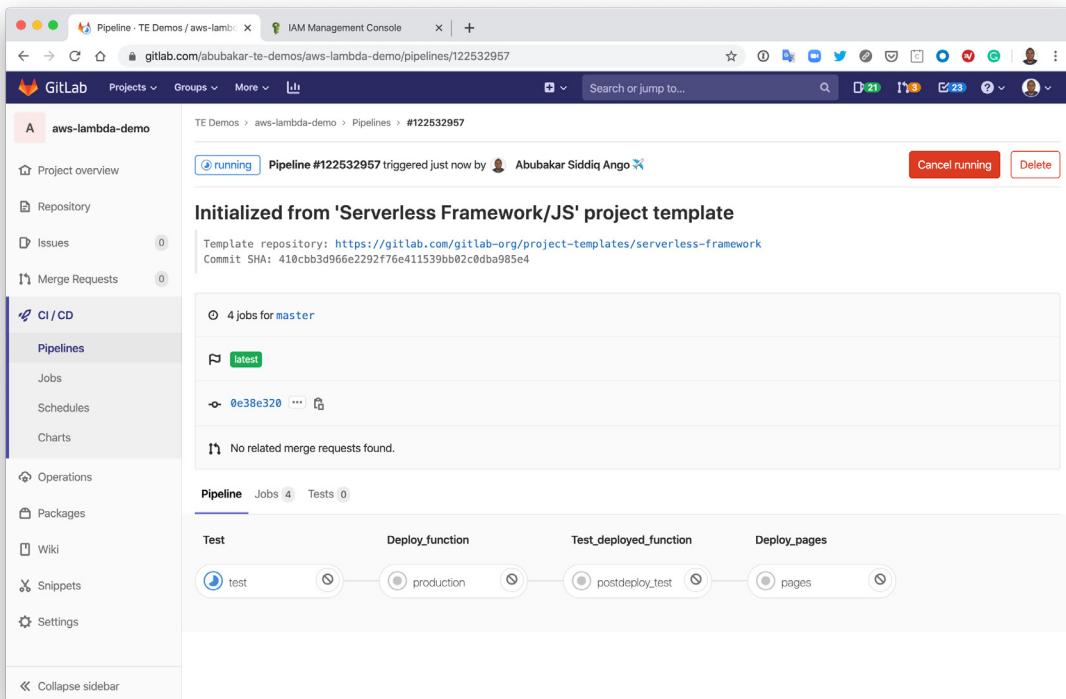
The project sample contains a `.gitlab-ci.yml` file. Changes made to the repository, or when a pipeline is initiated, four CI jobs (test, production, postdeploy_test and pages) are run as shown below:



```

image: node:latest
variables:
  AWS_REGION: us-east-1
stages:
  - test
  - deploy_function
  - test_deployed_function
  - deploy_pages
test:
  stage: test
  script:
    - npm install
    - npm test
production:
  stage: deploy_function
  before_script:
    - $FAILURE_MESSAGE="Must define \$AWS_ACCESS_KEY_ID and \$AWS_SECRET_ACCESS_KEY. Add keys to $CI_PROJECT_URL/-/settings/ci_cd"
    - test -z "$AWS_SECRET_ACCESS_KEY" && echo $FAILURE_MESSAGE && exit 1
    - test -z "$AWS_ACCESS_KEY_ID" && echo $FAILURE_MESSAGE && exit 1
  script:
    - npm install
    - npm run deploy --stage production --verbose
  environment: production
only:
  - master
artifacts:
  paths:
    - stack.json
    expire_in: 2 weeks
postdeploy_test:
  stage: test_deployed_function
  only:
    - master
  script:
    - npm install
    - STACK_JSON_FILE=stack.json npm test featureTests
pages:
  stage: deploy_pages
  script:
    - cp stack.json ./public/stack.json
  environment: production_pages
  only:
    - master
  artifacts:
    paths:
      - ./public/

```



A aws-lambda-demo

Project overview Repository Issues Merge Requests CI / CD Pipelines Jobs Schedules Operations Packages Analytics Wiki Snippets Settings

Initialized from 'Serverless Framework/JS' project template

Template repository: <https://gitlab.com/gitlab-org/project-templates/serverless-framework>
Commit SHA: 410cbb3d966e2292f76e411539bb02c0dba985e4

Pipeline #122532957 triggered 22 hours ago by Abubakar Siddiq Ango

9 jobs for master in 7 minutes and 4 seconds (queued for 162 minutes and 13 seconds)

latest

0e38e320

No related merge requests found.

Pipeline Jobs 9 Tests 0

Test	Deploy_function	Test_deployed_function	Deploy_pages	Deploy
test	production	postdeploy_test	pages	pages:deploy

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

A aws-lambda-demo

Project overview Repository Issues Merge Requests CI / CD Pipelines Jobs Schedules Operations Packages Analytics Wiki Snippets Settings

```

82 CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Deployment - ApiGatewayDeployment1583170045317
83 CloudFormation - CREATE_COMPLETE - AWS::ApiGateway::Deployment - ApiGatewayDeployment1583170045317
84 CloudFormation - CREATE_COMPLETE - AWS::Lambda::Permission - HelloLambdaPermissionApiGateway
85 CloudFormation - UPDATE_COMPLETE_CLEANUP_IN_PROGRESS - AWS::CloudFormation::Stack - gitlab-example-production
86 CloudFormation - UPDATE_COMPLETE - AWS::CloudFormation::Stack - gitlab-example-production
87 Serverless: Stack update finished...
88 Service Information
89 service: gitlab-example
90 stage: production
91 region: us-east-1
92 stack: gitlab-example-production
93 resources: 12
94 api keys:
95 None
96 endpoints:
97 GET - https://8sxglpasw9.execute-api.us-east-1.amazonaws.com/production/hello
98 functions:
99 hello: gitlab-example-production-hello
100 layers:
101 None
102 Stack Outputs
103 HelloLambdaFunctionQualifiedArn: arn:aws:lambda:us-east-1:206038703961:function:gitlab-example-production-hello:1
104 ServiceEndpoint: https://8sxglpasw9.execute-api.us-east-1.amazonaws.com/production
105 ServerlessDeploymentBucketName: gitlab-example-production-serverlessdeploymentbucket-1flxdolsfj5ja
106 Serverless: Run the "serverless" command to setup monitoring, troubleshooting and testing.
107 Serverless: Stack Output processed with handler: src/handler.hello
108 Serverless: Stack Output saved to file: stack.json
112 Uploading artifacts...
113 stack.json: found 1 matching files
114 Uploading artifacts to coordinator... ok          id=456475529 responseStatus=201 Created token=689riPPR
116 Job succeeded
  
```

Collapsible sidebar

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

The screenshot shows the Postman application interface. At the top, there's a header bar with 'GET Lambda' and a 'No Environment' button. Below the header, the title 'Lambda' is displayed with a 'Comments (0)' link. The main area shows a 'GET' request to the URL <https://8sxglpasw9.execute-api.us-east-1.amazonaws.com/production/hello>. A 'Send' button is visible on the right. Below the URL input, tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are present, with 'Params' being the active tab. Under 'Query Params', there's a table with one row: 'Key' (Value) and 'Value' (Description). Below the table, tabs for 'Body', 'Cookies', 'Headers (12)', and 'Test Results' are shown, with 'Body' being the active tab. On the right, status information 'Status: 200 OK Time: 1005ms Size: 580 B' is displayed. The 'Body' section shows a JSON response with four numbered lines: 1. {}, 2. "message": "AWS Lambda using GitLab", 3. "params": null, 4. }. The 'JSON' dropdown menu is open, and a copy icon is visible.

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:

```
abubakar_gitlab@Abubakars-MacBook-Pro: ~
🍺 /usr/local/Cellar/sqlite/3.31.1: 11 files, 4MB
==> Installing aws/tap/aws-sam-cli dependency: python
==> Downloading https://homebrew.bintray.com/bottles/python-3.7.6_1.catalina.bottle.tar.gz
==> Downloading from https://akamai.bintray.com/38/3871ef8b53270576c46489ae397f245b84772c4050
#####
100.0%
==> Pouring python-3.7.6_1.catalina.bottle.tar.gz
==> /usr/local/Cellar/python/3.7.6_1/bin/python3 -s setup.py --no-user-cfg install --force --
==> /usr/local/Cellar/python/3.7.6_1/bin/python3 -s setup.py --no-user-cfg install --force --
==> /usr/local/Cellar/python/3.7.6_1/bin/python3 -s setup.py --no-user-cfg install --force --
==> Caveats
Python has been installed as
/usr/local/bin/python3

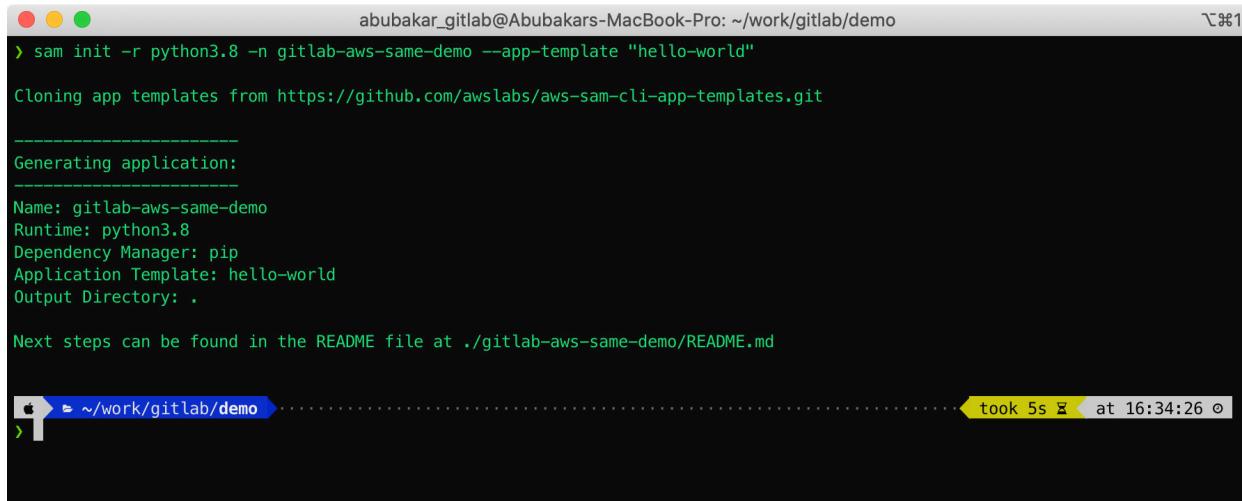
Unversioned symlinks `python`, `python-config`, `pip` etc. pointing to
`python3`, `python3-config`, `pip3` etc., respectively, have been installed into
/usr/local/opt/python/libexec/bin

You can install Python packages with
  pip3 install <package>
They will install into the site-package directory
  /usr/local/lib/python3.7/site-packages

See: https://docs.brew.sh/Homebrew-and-Python
==> Summary
🍺 /usr/local/Cellar/python/3.7.6_1: 3,977 files, 61MB
==> Installing aws/tap/aws-sam-cli
==> Downloading https://github.com/awslabs/aws-sam-cli/releases/download/v0.43.0//aws-sam-cli
==> Downloading from https://github-production-release-asset-2e65be.s3.amazonaws.com/92205085
#####
100.0%
==> Pouring aws-sam-cli-0.43.0.sierra.bottle.tar.gz
🍺 /usr/local/Cellar/aws-sam-cli/0.43.0: 3,975 files, 63.2MB
==> `brew cleanup` has not been run in 30 days, running now...
Removing: /Users/abubakar_gitlab/Library/Caches/Homebrew/dart--2.7.0.zip... (205.7MB)
Removing: /Users/abubakar_gitlab/Library/Caches/Homebrew/gmp--6.1.2_2.catalina.bottle.1.tar.gz... (996.4KB)
Removing: /Users/abubakar_gitlab/Library/Caches/Homebrew/helm--3.0.2.catalina.bottle.tar.gz... (12.1MB)
```



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

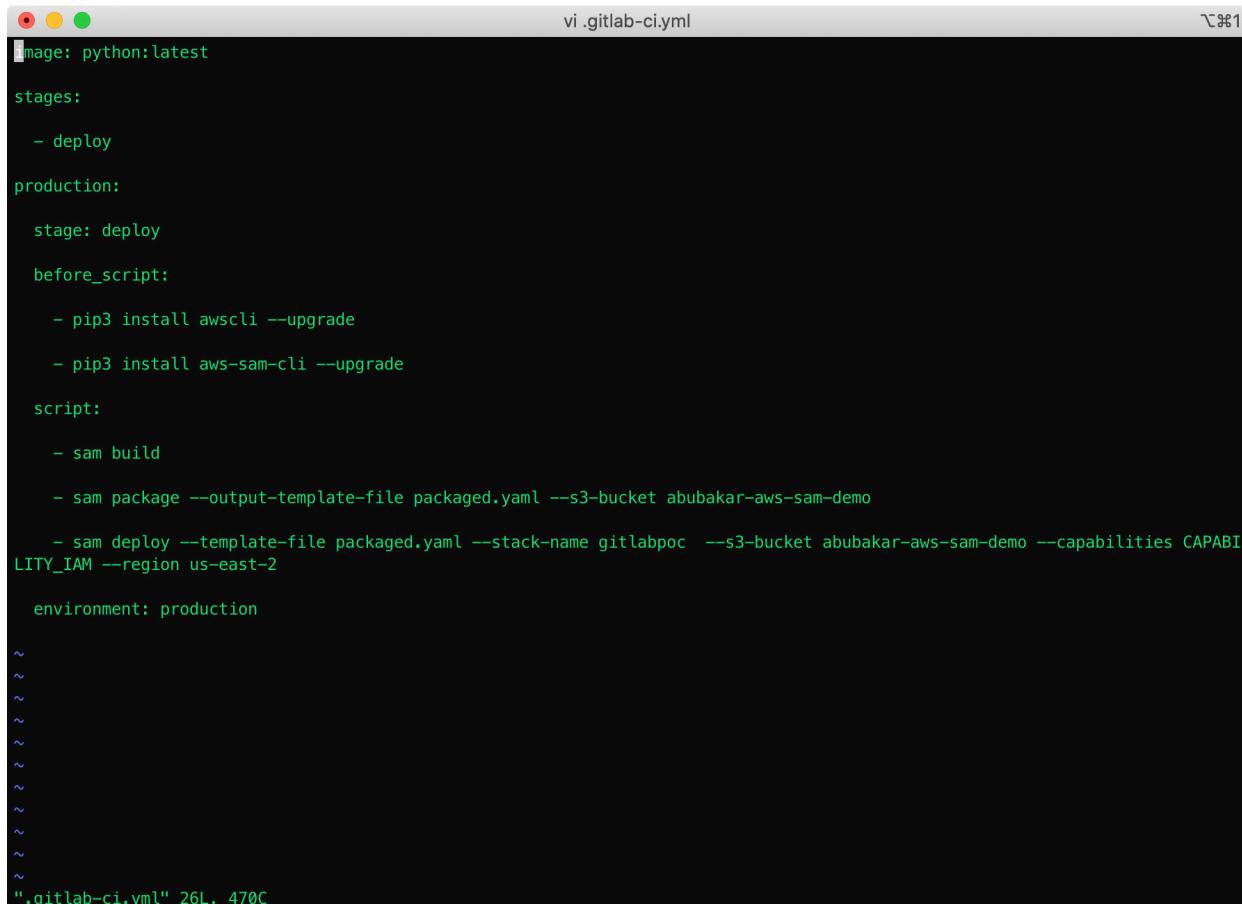


```
abubakar_gitlab@Abubakars-MacBook-Pro: ~/work/gitlab/demo
> sam init -r python3.8 -n gitlab-aws-same-demo --app-template "hello-world"
Cloning app templates from https://github.com/awslabs/aws-sam-cli-app-templates.git
-----
Generating application:
-----
Name: gitlab-aws-same-demo
Runtime: python3.8
Dependency Manager: pip
Application Template: hello-world
Output Directory: .

Next steps can be found in the README file at ./gitlab-aws-same-demo/README.md

> took 5s ✘ at 16:34:26
```

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



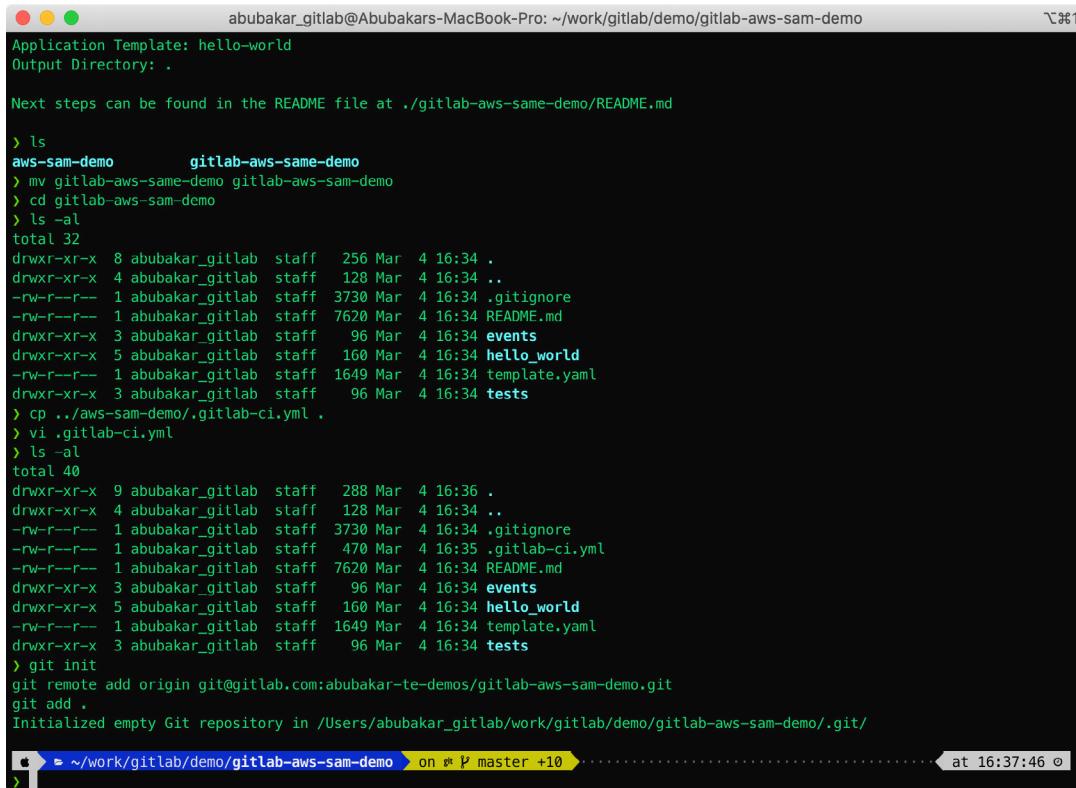
```
vi .gitlab-ci.yml
image: python:latest

stages:
- deploy

production:
  stage: deploy
  before_script:
    - pip3 install awscli --upgrade
    - pip3 install aws-sam-cli --upgrade
  script:
    - sam build
    - sam package --output-template-file packaged.yaml --s3-bucket abubakar-aws-sam-demo
    - sam deploy --template-file packaged.yaml --stack-name gitlabpoc --s3-bucket abubakar-aws-sam-demo --capabilities CAPABILITY_IAM --region us-east-2
  environment: production

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~.gitlab-ci.yml" 26L, 470C
```

Next, we run git init and push to GitLab:



```
abubakar_gitlab@Abubakars-MacBook-Pro: ~/work/gitlab/demo/gitlab-aws-sam-demo
```

```
Application Template: hello-world
Output Directory: .

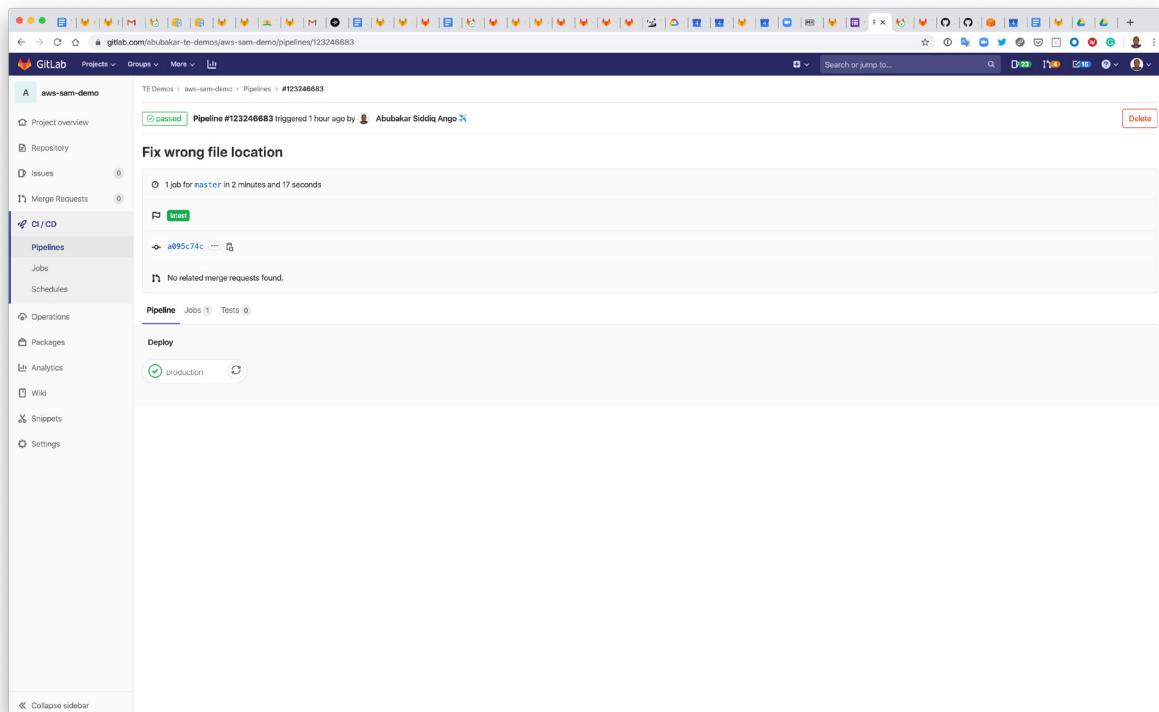
Next steps can be found in the README file at ./gitlab-aws-same-demo/README.md

> ls
aws-sam-demo      gitlab-aws-same-demo
> mv gitlab-aws-same-demo gitlab-aws-sam-demo
> cd gitlab-aws-sam-demo
> ls -al
total 32
drwxr-xr-x  8 abubakar_gitlab staff  256 Mar  4 16:34 .
drwxr-xr-x  4 abubakar_gitlab staff  128 Mar  4 16:34 ..
-rw-r--r--  1 abubakar_gitlab staff 3730 Mar  4 16:34 .gitignore
-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
> cp ../*aws-sam-demo/.gitlab-ci.yml .
> vi .gitlab-ci.yml
> ls -al
total 40
drwxr-xr-x  9 abubakar_gitlab staff  288 Mar  4 16:36 .
drwxr-xr-x  4 abubakar_gitlab staff  128 Mar  4 16:34 ..
-rw-r--r--  1 abubakar_gitlab staff 3730 Mar  4 16:34 .gitignore
-rw-r--r--  1 abubakar_gitlab staff 470 Mar  4 16:35 .gitlab-ci.yml
-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/

```

at 16:37:46 o

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



The screenshot shows a GitLab pipeline job named 'production' for a project 'gitlab-aws-sam-demo'. The job log displays the following content:

```

175 Region : us-east-2
176 Confirm changeset : False
177 Deployment s3 bucket : abubakar-aws-sam-demo
178 Capabilities : ["CAPABILITY_IAM"]
179 Parameter overrides : {}
180 Initiating deployment
181
182 Uploading to 4cad90319baef72e2fc9138a8b76df.template 1069 / 1069.0 (100.00%)
183 Waiting for changeset to be created...
184 CloudFormation stack changeset
185
186 Operation LogicalResourceId ResourceType
187
188 * Modify AWS::Lambda::Function AWS::Lambda::Function
189 * Modify ServerlessRestApi AWS::ApiGateway::RestApi
190
191 Changeset created successfully. arn:aws:cloudformation:us-east-2:206038703961:changeSet/samcli-deploy158336399/4573a982-18a5-4c88-b1e6-6e351b0f3be2
192 2020-03-04 15:40:04 - Waiting for stack create/update to complete
193 CloudFormation events from changeset
194
195 ResourceStatus ResourceType LogicalResourceId ResourceStatusReason
196
197 UPDATE_IN_PROGRESS AWS::Lambda::Function HelloWorldFunction -
198 UPDATE_COMPLETE AWS::Lambda::Function HelloWorldFunction -
199 UPDATE_COMPLETE_CLEANUP AWS::CloudFormation::Interface gitalbpc -
200 P_IN_PROGRESS tack -
201 UPDATE_COMPLETE AWS::CloudFormation::Interface gitalbpc -
202
203
204 CloudFormation outputs from deployed stack
205
206 Outputs
207
208 Key HelloWorldFunctionIamRole
209 Description Implicit IAM Role created for Hello World function
210 Value arn:aws:iam::206038703961:role/gitalbpc-
211 HelloWorldFunctionRole-1XK6T1Q0XK76
212 Key HelloWorldApi
213 Description API Gateway endpoint URL for Prod stage for Hello World function
214 Value https://xlylz8bls7.execute-api.us-east-2.amazonaws.com/Prod/hello/
215
216 Description Hello World Lambda Function ARN
217 Value arn:aws:lambda:us-east-2:206038703961:function:gitalbpc-
218 HelloWorldFunction-1P92KZJ02YK7
219
220 Successfully created/updated stack - gitalbpc in us-east-2
221 Job succeeded

```

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
-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"}<

```

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.

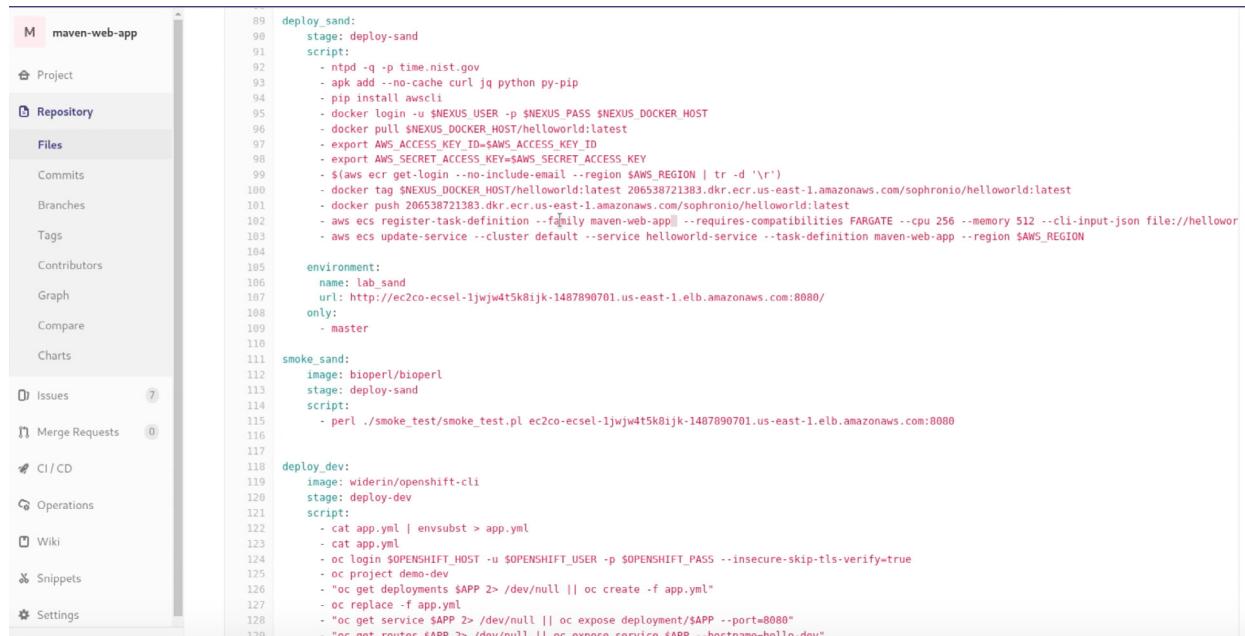
AWS Fargate is a serverless compute engine for containers that works with both [Amazon Elastic Container Service \(ECS\)](#) and [Amazon Elastic Kubernetes Service \(EKS\)](#). Fargate primarily removes the need to provision and manage servers and allows developers to specify resources for each application.

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:



The screenshot shows a GitLab project page for 'maven-web-app'. On the left, there's a sidebar with various project management links like Project, Repository, Files, Commits, Branches, Tags, Contributors, Graph, Compare, Charts, Issues (7), Merge Requests (0), CI / CD, Operations, Wiki, Snippets, and Settings. The main area displays a pipeline configuration file with syntax highlighting for YAML and shell commands. The file includes sections for 'deploy_sand', 'environment', 'smoke_sand', 'deploy_dev', and 'deploy_staging'. It uses AWS CLI and Docker commands to handle deployment tasks.

```
89 deploy_sand:
90   stage: deploy-sand
91   script:
92     - ntpd -q -p time.nist.gov
93     - apk add -no-cache curl jq python py-pip
94     - pip install awscli
95     - docker login -u $NEXUS_USER -p $NEXUS_PASS $NEXUS_DOCKER_HOST
96     - docker pull $NEXUS_DOCKER_HOST/helloworld:latest
97     - export AWS_ACCESS_KEY_ID=$AWS_ACCESS_KEY_ID
98     - export AWS_SECRET_ACCESS_KEY=$AWS_SECRET_ACCESS_KEY
99     - $(aws ecr get-login --no-include-email --region $AWS_REGION | tr -d '\r')
100    - docker tag $NEXUS_DOCKER_HOST/helloworld:latest 206538721383.dkr.ecr.us-east-1.amazonaws.com/sophronio/helloworld:latest
101    - docker push 206538721383.dkr.ecr.us-east-1.amazonaws.com/sophronio/helloworld:latest
102    - aws ecs register-task-definition --family maven-web-app --requires-compatibilities FARGATE --cpu 256 --memory 512 --cli-input-json file://helloworld/task-definition.json
103    - aws ecs update-service --cluster default --service helloworld-service --task-definition maven-web-app --region $AWS_REGION
104
105 environment:
106   name: lab_sand
107   url: http://ec2co-ecsel-1jwjw4t5k8ijk-1487890701.us-east-1.elb.amazonaws.com:8080/
108   only:
109     - master
110
111 smoke_sand:
112   image: bioperl/bioperl
113   stage: deploy-sand
114   script:
115     - perl ./smoke_test/smoke_test.pl ec2co-ecsel-1jwjw4t5k8ijk-1487890701.us-east-1.elb.amazonaws.com:8080
116
117
118 deploy_dev:
119   image: widerin/openshift-cli
120   stage: deploy-dev
121   script:
122     - cat app.yml | envsubst > app.yaml
123     - cat app.yaml
124     - oc login $OPENSHIFT_HOST -u $OPENSHIFT_USER -p $OPENSHIFT_PASS --insecure-skip-tls-verify=true
125     - oc project demo-dev
126     - *oc get deployments $APP 2> /dev/null || oc create -f app.yaml*
127     - oc replace -f app.yaml
128     - *oc get service $APP 2> /dev/null || oc expose deployment/$APP --port=8080*
129     - *oc get routes $APP 2> /dev/null || oc expose service $APP --hostname=helloworld.dev*
```

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.

TE Demos > Kubernetes

Add a Kubernetes cluster integration

Adding a Kubernetes cluster to your group will automatically share the cluster across all your projects. Use review apps, deploy your applications, and easily run your pipelines for all projects using the same cluster.

[Learn more about group Kubernetes clusters](#)

Create new cluster on EKS Add existing cluster

Create cluster on

 Amazon EKS  Google GKE

Authenticate with Amazon Web Services

You must grant access to your organization's AWS resources in order to create a new EKS cluster. To grant access, create a provision role using the account and external ID below and provide us the ARN.

Account ID External ID



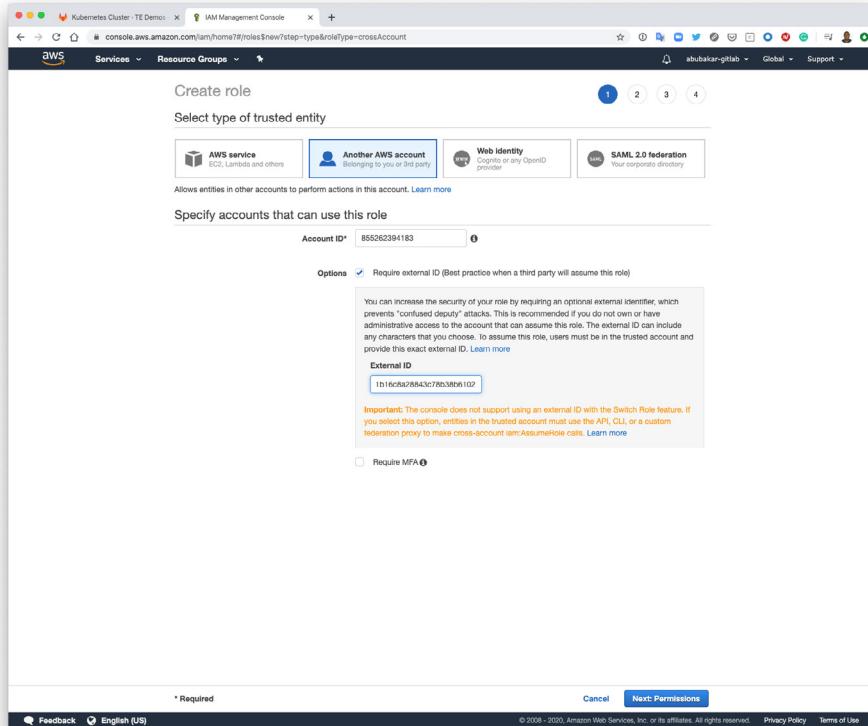
Create a provision role on [Amazon Web Services](#) using the account and external ID above. [More information](#)

Provision Role ARN

The Amazon Resource Name (ARN) associated with your role. If you do not have a provision role, first create one on [Amazon Web Services](#) using the above account and external IDs. [More information](#)

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

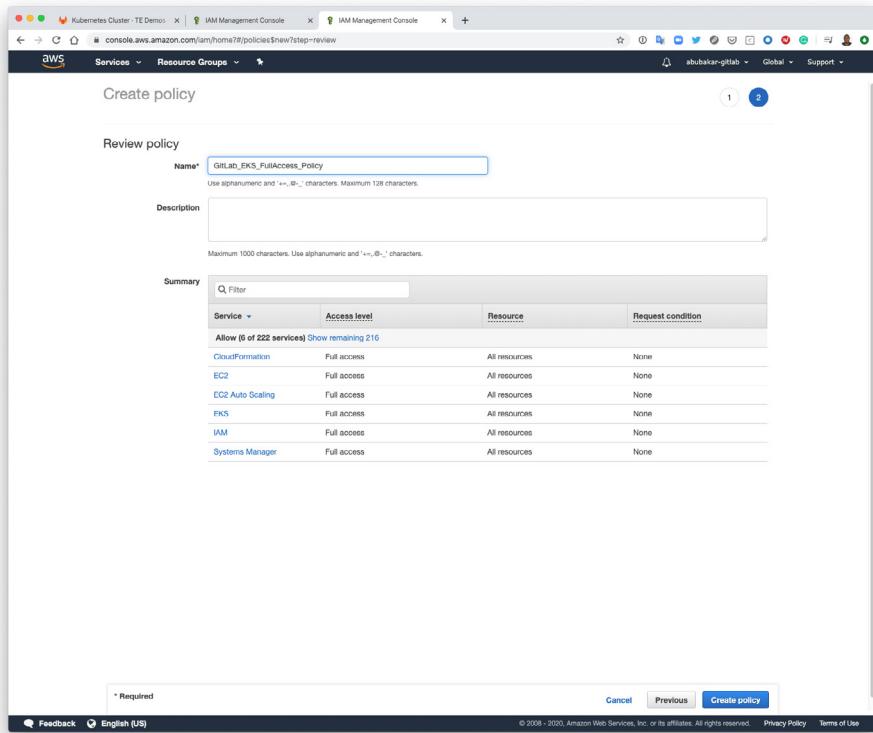
- Click on the “Another AWS account” tab and provide the Account and External ID obtained from GitLab and click Next to set permissions as shown below:



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			
<input type="text"/> 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.

Create role

1 2 3 4

Select type of trusted entity

AWS service Another AWS account Web identity SAML 2.0 federation

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Or select a service to view its use cases

API Gateway	CodeDeploy	EMR	KMS	RoboMaker
AWS Backup	CodeGuru	ElastiCache	Kinesis	S3
AWS Chatbot	CodeStar Notifications	Elastic Beanstalk	Lambda	SMS
AWS Support	Comprehend	Elastic Container Service	Lex	SNS
Amplify	Config	Elastic Transcoder	License Manager	SWF
AppStream 2.0	Connect	Elastic Load Balancing	Machine Learning	SageMaker
AppSync	DMS	Forecast	Macie	Security Hub
Application Auto Scaling	Data Lifecycle Manager	Global Accelerator	MediaConvert	Service Catalog
Application Discovery Service	Data Pipeline	Glue	Migration Hub	Step Functions
Batch	DataSync	Greengrass	OpsWorks	Storage Gateway
Chime	DeepLens	GuardDuty	Personalize	Textract
CloudFormation	Directory Service	Health Organizational View	QLDB	Transfer
CloudHSM	DynamoDB	IAM Access Analyzer	RAM	Trusted Advisor
CloudTrail	EC2	Inspector	RDS	VPC
CloudWatch Application Insights	EC2 - Fleet	IoT	Redshift	WorkLink
CloudWatch Events	EC2 Auto Scaling	IoT Things Graph	Rekognition	WorkMail
CodeBuild	EKS			

Select your use case

EKS
Allows EKS to manage clusters on your behalf.

EKS - Fargate pod
Allows access to other AWS service resources that are required to run Amazon EKS pods on AWS Fargate.

EKS - Fargate profile
Allows EKS to run Fargate tasks.

EKS - Nodegroup
Allow EKS to manage nodegroups on your behalf.

* Required

Cancel **Next: Permissions**

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.

The screenshot shows the AWS IAM Management Console interface. On the left, there's a navigation sidebar with options like 'Dashboard', 'Access management', 'Groups', 'Users', 'Roles' (which is selected), 'Policies', 'Identity providers', and 'Account settings'. The main area displays the details of a role named 'GitLab_EKS_Role'. Key information shown includes:

- Role ARN:** arn:aws:iam::206038703961:role/GitLab_EKS_Role
- Role description:** Edit
- Instance Profile ARN:** [Edit](#)
- Path:** /
- Creation time:** 2020-02-23 07:55 UTC+0100
- Last activity:** Not accessed in the tracking period
- Maximum CLI/API session duration:** 1 hour [Edit](#)
- Give this link to users who can switch roles in the console:** https://signin.aws.amazon.com/switchrole?roleName=GitLab_EKS_Role&account=206038703961

The 'Permissions' tab is active, showing one policy applied:

- Policy name:** GitLab_EKS_FullAccess_Policy
- Policy type:** Managed policy

Other tabs available include 'Trust relationships', 'Tags', 'Access Advisor', and 'Revoke sessions'.

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."

The screenshot shows the 'Add a Kubernetes cluster integration' section of the GitLab interface. It has two main tabs: 'Create new cluster on EKS' (selected) and 'Add existing cluster'. Below this, there are two options: 'Amazon EKS' and 'Google GKE'. Under 'Amazon EKS', there is a section titled 'Authenticate with Amazon Web Services' with instructions: 'You must grant access to your organization's AWS resources in order to create a new EKS cluster. To grant access, create a provision role using the account and external ID below and provide us the ARN.' It includes fields for 'Account ID' (855262394183) and 'External ID' (94a9f5703b0c7d7b31b16c8a28843c78b38b6102), and a note about creating a provision role on AWS. There is also a 'Provision Role ARN' field and a note about its creation. At the bottom is a 'Authenticate with AWS' button.

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.

The screenshot shows the 'Add a Kubernetes cluster integration' page on the GitLab interface. The left sidebar shows the 'Kubernetes' section is selected. The main form is titled 'Create new cluster on EKS' and includes options for 'Amazon EKS' and 'Google GKE'. The 'Amazon EKS' section is active, showing fields for 'Kubernetes cluster name' (set to 'gitlab-eks-us-east'), 'Environment scope' (set to '*'), 'Kubernetes version' (set to '1.14'), 'Service role' (set to 'GitLab_EKS_Service_Role'), 'Region' (set to 'us-east-2'), 'Key pair name' (set to 'abubakar'), 'VPC' (set to 'vpc-a4ef20cf'), 'Subnets' (listing three subnets: 'subnet-d9db4f95', 'subnet-7d38cf16', 'subnet-2fd3fe55'), 'Security group' (set to 'default'), 'Instance type' (set to 't2.large'), and 'Number of nodes' (set to '3'). A checkbox for 'GitLab-managed cluster' is checked. At the bottom is a green 'Create Kubernetes cluster' button.

- 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 for 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.

The screenshot shows the GitLab interface for managing a Kubernetes cluster. The top navigation bar includes tabs for 'Kubernetes Cluster - TE Demos', 'Amazon EKS Service IAM Role', and 'IAM Management Console'. The main content area is titled 'gitlab-eks-us-east' under 'Kubernetes Clusters'. A green success message states 'gitlab-eks-us-east was successfully created.' Below this, there are sections for 'Integration status' (with a toggle switch), 'Environment scope' (a dropdown menu), and 'Base domain' (a text input field). A 'Save changes' button is present. The 'Applications' section lists several pre-defined applications with 'Install' buttons:

- Helm Tiller**: Described as streamlining installing and managing Kubernetes applications. An 'Install' button is shown.
- Ingress**: Described as routing requests to services based on host or path. An 'Install' button is shown.
- Cert-Manager**: Described as a native Kubernetes certificate management controller. An 'Install' button is shown.
- Prometheus**: Described as an open-source monitoring system. An 'Install' button is shown.
- GitLab Runner**: Described as connecting to the repository and executing CI/CD jobs. An 'Install' button is shown.
- Crossplane**: Described as enabling declarative provisioning of managed services. An 'Install' button is shown.
- JupyterHub**: Described as a multi-user Hub for Jupyter notebooks. An 'Install' button is shown.

At the bottom left, there is a sidebar with 'Collapse sidebar' and a user profile icon. The bottom right corner shows the URL 'abubakar (1).pm'.

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:

Clusters > my-cluster

Cluster : my-cluster

Get a detailed view of the resources on your cluster.

Status ACTIVE

Registered container instances 1

Pending tasks count 0 Fargate, 0 EC2

Running tasks count 0 Fargate, 1 EC2

Active service count 0 Fargate, 1 EC2

Draining service count 0 Fargate, 0 EC2

Services **Tasks** **ECS Instances** **Metrics** **Scheduled Tasks** **Tags** **Capacity Providers**

Create Update Delete Actions ▾

Last updated on March 4, 2020 3:42:39 PM (2m ago)

Filter in this page	Launch type	ALL	Service type	ALL	Last updated on March 4, 2020 3:42:39 PM (2m ago)				
<input type="checkbox"/> Service Name	Status	ACTIVE	Service type	REPLICAS	Task Definition	Desired tasks	Running tasks	Launch type	Platform version
<input type="checkbox"/> my-service	ACTIVE	REPLICAS	my-task-definition:10	1	0	EC2	-		

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:

The screenshot shows the AWS EC2 'Choose Instance Type' step. The 'General purpose' section is selected, displaying various instance types such as t2.nano, t2.micro, t2.small, t2.medium, t2.large, t2.xlarge, t2.2xlarge, t3.nano, t3.micro, t3.small, t3.medium, t3.large, t3.xlarge, t3.2xlarge, and t3a.nano. The t2.micro instance is highlighted with a green border. A modal window in the center shows a preview of the GitLab interface. At the bottom right, there are 'Cancel', 'Previous', and 'Review and Launch' buttons. The time '3:27' is visible at the bottom center of the modal.

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**

The screenshot shows the AWS Management Console homepage. In the center, there's a section titled "Launch a virtual machine" with a sub-section "With EC2". It says "2-3 minutes" and has a small icon of a server. To the right, there are other options: "Build a web app" (With Elastic Beanstalk, 6 minutes), "Build using virtual servers" (With Lightsail, 1-2 minutes), "Register a domain" (With Route 53, 3 minutes), "Connect an IoT device" (With AWS IoT, 5 minutes), "Start migrating to AWS" (With CloudEndure Migration, 1-2 minutes), "Start a development project" (With CodeStar, 5 minutes), and "Deploy a serverless microservice" (With Lambda, API Gateway, 2 minutes). On the right side of the page, there's a sidebar titled "Access resources on the go" with a link to the AWS Mobile App. Below it is a section titled "Explore AWS" with links to "AWS IQ", "Amazon DynamoDB", "Amazon GuardDuty", and "Free Digital Training". At the bottom right, there's a "Have feedback?" button.

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

The screenshot shows the "Choose AMI" step in the AWS Management Console. The user has searched for "gitlab 12.8" and found several results. The first result is "bitnami-gitlab-11.0.4-0-linux-debian-8-x86_64-ebi" with the ID "ami-012d884415155c9d". Below it are other results for GitLab EE 12.8.1, GitLab CE 12.8.0, and GitLab CE 12.8.1.1. Each result provides details like the AMI ID, description, root device type, and ENA status. The interface includes navigation buttons at the top and a sidebar on the left with filters for operating system, architecture, and root device type.

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

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how needs.

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPU, 2.5 GHz, Intel Xeon Family, 1 GB memory, EBS only)

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized Available	Network Performance
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate
General purpose	t2.micro <small>(Variable ECUs)</small>	1	1	EBS only	-	Low to Moderate
General purpose	t2.small	1	2	EBS only	-	Low to Moderate
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
General purpose	t2.large	2	8	EBS only	-	Low to Moderate
General purpose	t2.xlarge	4	16	EBS only	-	Moderate
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate
General purpose	t3.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.micro	2	1	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.small	2	2	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.medium	2	4	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.large	2	8	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.xlarge	4	16	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.2xlarge	8	32	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.micro	2	1	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.small	2	2	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.medium	2	4	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.large	2	8	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.xlarge	4	16	EBS only	Yes	Up to 5 Gigabit
General purpose	t3.2xlarge	8	32	EBS only	Yes	Up to 5 Gigabit

3:27

Cancel Previous Review and Launch

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

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click Launch to assign a key pair to your instance and complete the launch process.

AMI Details

GitLab EE 12.8.1 - ami-055a96145a81e76b6
Official GitLab EE 12.8.1 AMI: <https://about.gitlab.com/>
Root Device Type: ebs Virtualization type: hvm

Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GiB)	EBS-Optimized
t2.medium	Variable	2	4	EBS only	-

Security Groups

Security group name: launch-wizard-6
Description: launch-wizard-6 created 2020-02-27T13:31:16.346-08:00

Type	Protocol	Port Range	Source
This security group has no rules			

Select an existing key pair or create a new key pair

A key pair consists of a public key that AWS stores, and a private key file that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about managing existing key pairs from a public AMI.

Choose an existing key pair
Select a key pair gitlab-unest1-keypair
 gitlab-unest1-keypair
 I acknowledge that I have access to the selected private key file (gitlab-unest1-keypair.pem), and that without this file, I won't be able to log into my instance.

CANCEL 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.

Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	IPv4 Public IP	Key Name	Monitoring	Launch Time	Security Groups
i-04deaf97076b48d75	t2.medium	us-east-1c	running	2/2 checks ...	None	54.145.48.96	gitlab-user1...	disabled	February 27, 2020 at 1:51...	launch-wizard-4
i-0502168f95b...	t2.medium	us-east-1c	running	2/2 checks ...	None	34.229.17.190	gitlab-user1...	enabled	November 21, 2019 at 4:17...	kubeflow-nodestack-NodeSecurityGroup-7C
i-0cc0eefdf48...	t2.medium	us-east-1b	running	2/2 checks ...	None	54.152.185.166	gitlab-user1...	enabled	November 21, 2019 at 4:17...	kubeflow-nodestack-NodeSecurityGroup-7C
i-0e011a0...	t2.medium	us-east-1a	running	2/2 checks ...	None	34.205.87.91	gitlab-user1...	enabled	November 21, 2019 at 4:17...	kubeflow-nodestack-NodeSecurityGroup-7C
i-0e811d...	t2.medium	us-east-1d	running	2/2 checks ...	None	54.152.181.94	gitlab-user1...	disabled	October 7, 2019 at 12:49:22 ...	web-secgroup

Instance: i-04deaf97076b48d75 Public DNS: ec2-54-145-48-96.compute-1.amazonaws.com

Description	Status Checks	Monitoring	Tags
Instance ID: i-04deaf97076b48d75 Instance state: running Instance type: t2.medium Finding: Opt-in to AWS Compute Optimizer for recommendations. Learn more Private DNS: ip-172-31-24-78.ec2.internal Private IPs: 172.31.24.78 Secondary private IPs: VPC ID: vpc-0db1160045095304 Subnet ID: subnet-03b0aa9722bc1bb Network interfaces: eth0 Source/dest. check: True T/T/T Unlimited: Disabled EB5-optimized: False Root device type: ebs			
Public DNS (IPv4): ec2-54-145-48-96.compute-1.amazonaws.com IPv4 Public IP: 54.145.48.96 IPv6 (Ips): - Elastic IPs: Availability zone: us-east-1c Security groups: launch-wizard-4, view inbound rules, view outbound rules Scheduled events: No scheduled events AMI ID: GitLab EE 12.8.1 (ami-055a99145a81e762) Platform: - IAM role: - Key pair name: gitlab-user1-keypair Owner: 244717664470 Launch time: February 27, 2020 at 1:51:30 PM UTC-8 (less than one hour) Termination protection: False			

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.

The screenshot shows the AWS Marketplace search results for 'gitlab'. The search bar at the top contains 'gitlab'. Below it, there are several search filters: 'Quick Start (0)', 'My AMIs (0)', 'AWS Marketplace (12)', 'Community AMIs (743)', 'Categories' (with 'All Categories', 'Infrastructure Software (3)', 'DevOps (10)', 'Business Applications (2)'), 'Operating System' (with 'All Linux/Unix' (8), 'Amazon Linux (2)', 'Debian (2)', 'CentOS (1)', 'Other (1)'), 'Software Free Trial' (with 'Free Trial (2)'), 'Software Pricing Plans' (with 'Hourly (9)', 'Annual (8)', 'Free (2)', 'Bring Your Own License (1)'), 'Support' (with 'Product Support Connection (1)'), and 'Region' (with 'Current Region (12)', 'All Regions (208)').

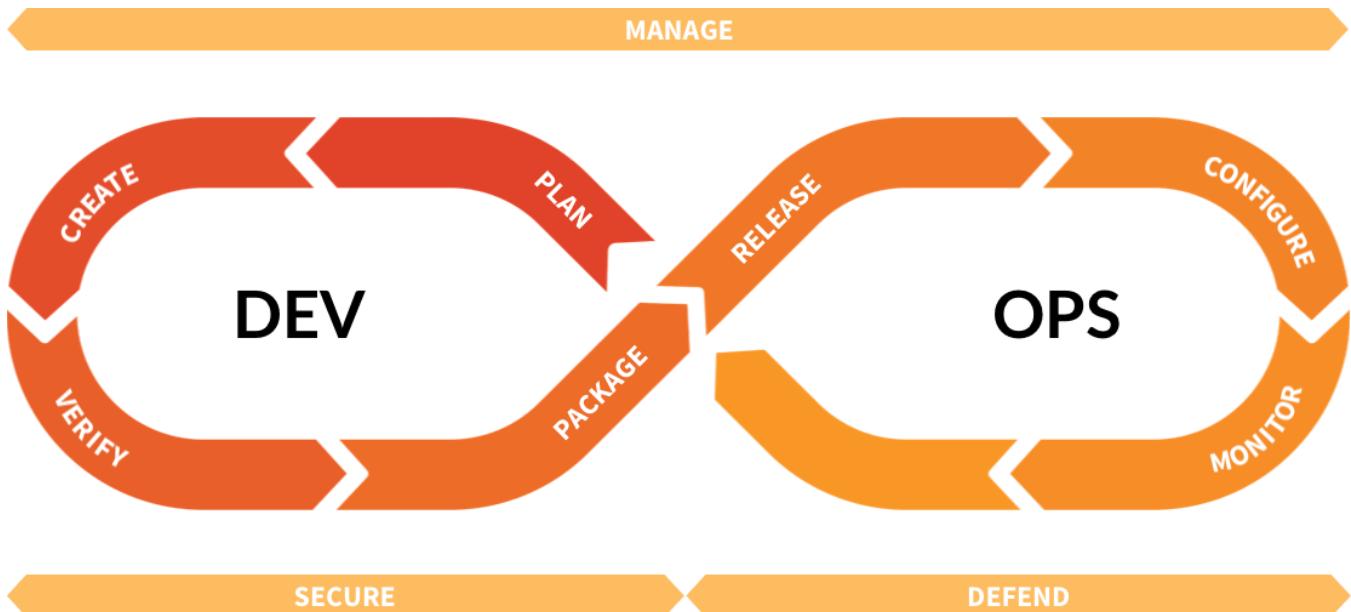
The main list displays the following results:

- GitLab Community Edition**: Free for eligible. Rating: ★★★★☆ (1). Description: GitLab CE 12.8.1 Previous versions | By GitLab. Last updated: 2/24/20. Includes: Git repository management, issue tracking, code review, an IDE, activity streams, wikis, and more.
- GitLab CE Certified by Bitnami**: Rating: ★★★★☆ (14). Description: GitLab 12.8.1-0 on Ubuntu 18.04 Previous versions | By Bitnami. Last updated: 2/25/20. Up-to-date and secure image. GitLab CE is an open source, cloud-based Git repository, and version control system. It allows DevOps teams to cover the full software development lifecycle from a single application.
- GitLab Ultimate - 5 User Pack**: Rating: ★★★★☆ (1). Description: GitLab Ultimate 12.8.1 Previous versions | By GitLab. Starting from \$1.07/hr or from \$5,940.00/year (up to 51% savings) for software + AWS usage fees. Includes: GitLab is the first single application for software development, security, and operations that enables Concurrent DevOps, making the software lifecycle 200% faster and radically improving the speed of business.
- GitLab on Ubuntu**: Rating: ★★★★☆ (1). Description: GitLab-CE12.8.1-ubuntu18.04 | By Veloce9. Starting from \$0.01/hr or from \$40.00/year (up to 51% savings) for software + AWS usage fees.
- GitLab CE powered by Classmethod**: Rating: ★★★★☆ (3). Description: GitLab-CE12.8.1-ubuntu18.04 | By Classmethod. Starting from \$0.01 to \$0.01/hr for software + AWS usage fees.
- Best Project Management Package**: Rating: ★★★★☆ (3). Description: Inzat Project Management 2.0 Previous versions | By Inzat. Starting from \$0.01/hr or from \$69.00/year (up to 43% savings) for software + AWS usage fees.

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.

