Cloud Computing

SE Master

CI / CD Cloud Provider Overview

Authors:
Andreas Roither, Markus Seiberl

January 28, 2021

Contents

1 Introduction

Each section explains components that are used to build the CI / CD pipeline for each provider and answers some questions about integrations and pricing. The complete pipeline step-by-step instructions are at the end of each section:

- AWS section 2.10
- Alibaba section 3.10
- Azure section 4.10
- Google section 5.9

An evaluation and cost comparison outlines the differences between the 4 different provider and concludes this paper.

2 AWS

This section describes the CI / CD process for Amazon Web Services.

2.1 Integration with different Version Control Systems

There are currently 4 supported ways of VCS integrations.

- Amazon S3
- AWS CodeCommit
- Github and Github Enterprise
- Bitbucket

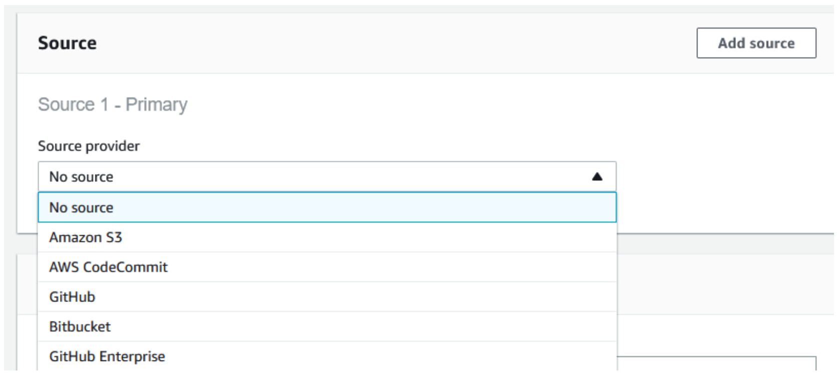


Figure 1: Different VCS integrations

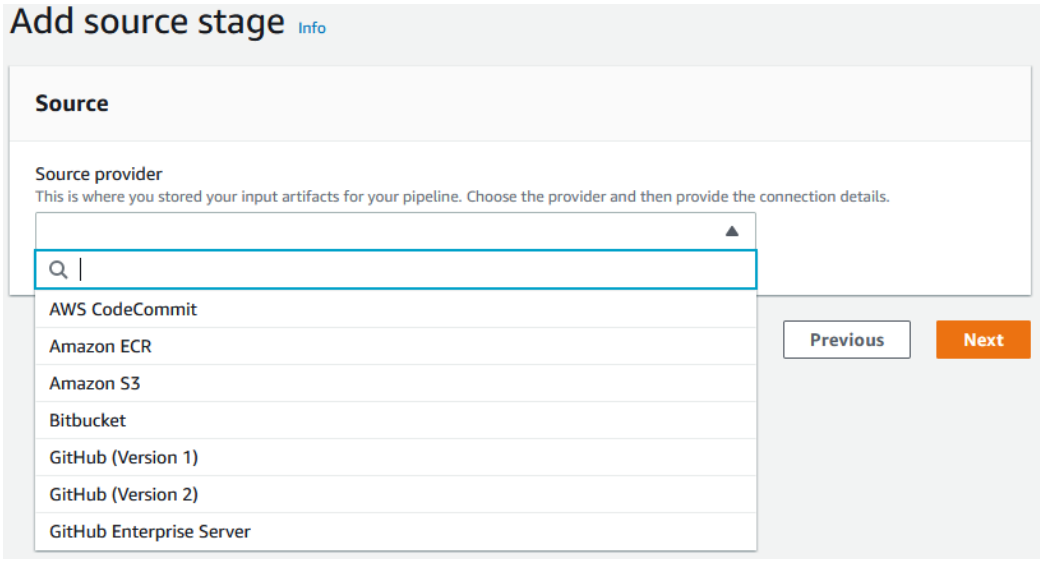
2.2 Cloud Build

Amazon offers two services that can be used to build and upload containers: CodeBuild and ECR (Elastic Container Registry).
Similar to Gcloud, GitLab or even Github actions can also be used to start builds. Possible alternatives are:

- AWS console and CI script that pushes changes
- AWS Fargate with coordinators that push into an AWS S3 bucket which triggers a cloud build
- AWS lambda function that pushes into a bucket

2.3 Pipelines

AWS CodePipeline can be used to create a pipeline for building, testing and deployment. The two most important sections, input and output are presented here beforehand. Different sources can be used for a pipeline:



Add source stage Info

Source

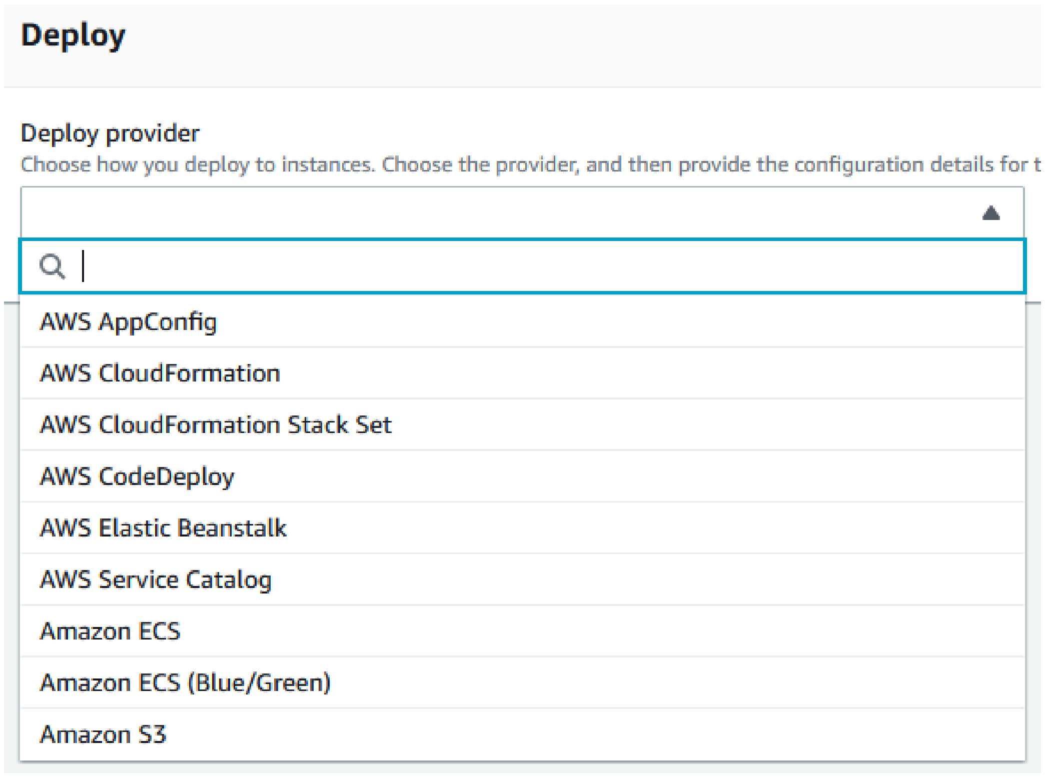
Source provider

This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

- AWS CodeCommit
- Amazon ECR
- Amazon S3
- Bitbucket
- GitHub (Version 1)
- GitHub (Version 2)
- GitHub Enterprise Server

Figure 2: CodePipeline source provider

Different deployment options:


Deploy

Deploy provider

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for t

Figure 3: CodePipeline deployment provider

As there are a number of different ways to create a CodePipeline in AWS (different end deployments), we will focus on Elastic Beanstalk as an example. The CodePipeline will be explained in detail later in section 2.10.

2.4 Deployment Strategies

To deploy our containerized Rest-API, Elastic Beanstalk is used.

AWS Elastic Beanstalk provides several options for how deployments are processed, including deployment policies (All at once, Rolling, Rolling with additional batch, Immutable, and Traffic splitting) and options that let you configure the batch size and health check behaviour during deployments. By default, the environment uses all-at-once deployments.

The Application deployments section of the Rolling updates and deployments page has the following options for application deployments:

- All at once
  Deploy the new version to all instances simultaneously. All instances in the environment are out of service for a short time while the deployment occurs.

- Rolling
  Deploy the new version in batches. Each batch is taken out of service during the deployment phase, reducing your environment's capacity by the number of instances in a batch.

- Rolling with additional batch
  Deploy the new version in batches, but first, launch a new batch of instances to ensure full capacity during the deployment process.

- Immutable
  Deploy the new version to a fresh group of instances by performing an immutable update.

- Traffic splitting
  Deploy the new version to a fresh group of instances and temporarily split incoming client traffic between the existing application version and the new one.

The Rolling and Rolling with additional batch deployment policies allow additional configuration:

- Batch size
  The size of the set of instances to deploy in each batch. Choose Percentage to configure a percentage of the total number of EC2 instances in the Auto Scaling group (up to 100 percent), or choose Fixed to configure a fixed number of instances.

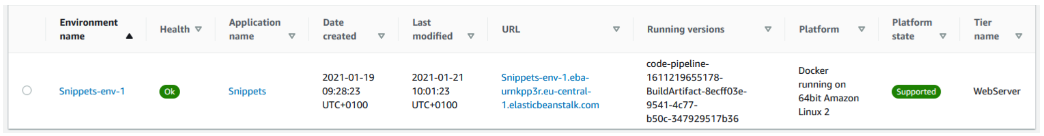
The Traffic splitting deployment policy has additional configuration options as well:

- Traffic split
  The initial percentage of incoming client traffic that Elastic Beanstalk shifts to environment instances running the new application version you're deploying.

- Traffic splitting evaluation time
  The time period, in minutes, that Elastic Beanstalk waits after an initial healthy deployment before proceeding to shift all incoming client traffic to the new application version that is currently deploying.

2.5 Environments

Amazon Elastic Beanstalk allows the creation of different applications; each application can have multiple environments, including a collection of AWS resources running an application version. Multiple environments can be deployed, each with a different version if desired.



	Environment name	Health	Application name	Date created	Last modified	URL	Running versions	Platform	Platform state	Tier name
○	Snippets-env-1	Ok	Snippets	2021-01-19 09:28:23 UTC+0100	2021-01-21 10:01:23 UTC+0100	Snippets-env-1.eba-urnkpp3r.eu-central-1.elasticbeanstalk.com	code-pipeline-1611219655178-BuildArtifact-8ecff03e-9541-4c77-b50c-347929517b36	Docker running on 64bit Amazon Linux 2	Supported	WebServer

Figure 4: Elastic Beanstalk Environment

2.6 Scaling

2.6.1 Elastic Beanstalk

Elastic Beanstalk allows auto-scaling for its instances. A minimum and maximum of instances can be set. Besides, metrics that are monitored can be used to determine if the environment's capacity is low:

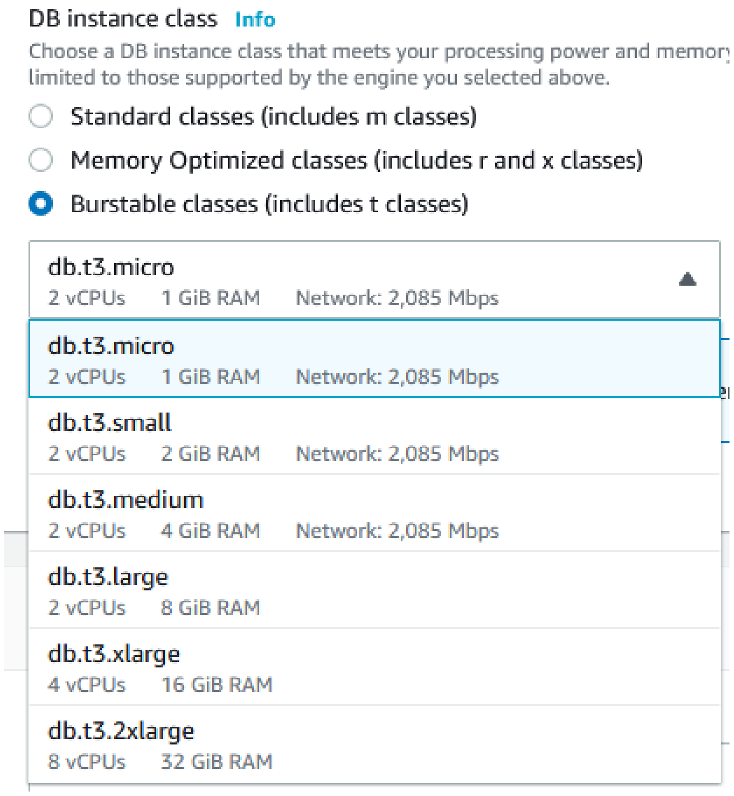
- CPUUtilization
- NetworkIn
- NetworkOut
- DiskWriteOps
- DiskReadBytes
- DiskWriteBytes
- Latency
- RequestCount

- HealthyHostCount
- UnhealthyHostCount
- TargetResponseTime

Furthermore, time-based scaling can also be enabled, where instances can be up- or down-scaled depending on the time.

2.6.2 RDS

Amazon RDS (Relational Database Service) instances can be scaled vertically after they have been created. By modifying the RDS instance, however, will result in downtime.



DB instance class Info

Choose a DB instance class that meets your processing power and memory
limited to those supported by the engine you selected above.

( ) Standard classes (includes m classes)
( ) Memory Optimized classes (includes r and x classes)
[✓ Burstable classes (includes t classes)]

db.t3.micro
2 vCPUs 1 GiB RAM Network: 2,085 Mbps

db.t3.micro	2 vCPUs	1 GiB RAM	Network: 2,085 Mbps
db.t3.small	2 vCPUs	2 GiB RAM	Network: 2,085 Mbps
db.t3.medium	2 vCPUs	4 GiB RAM	Network: 2,085 Mbps
db.t3.large	2 vCPUs	8 GiB RAM	
db.t3.xlarge	4 vCPUs	16 GiB RAM	
db.t3.2xlarge	8 vCPUs	32 GiB RAM	

Figure 5: RDS classes

RDS instances can also be scaled horizontally to reduce the load. By creating read replicas (read only) that are synchronized with the master, the load on a single instance can be lessened. Additionally, replicas can be placed into different regions.

2.7 Extensions

The AWS CodePipeline has pre-built plugins for source control, building, testing and deployment. Custom plugins can be created by registering a custom action that allows hooking custom servers into the pipeline by integrating the CodePipeline open-source agent with the custom servers. The CodePipeline Jenkins plugin can also be used to register existing build servers as a custom action.
Example of integrations:

- Source
  Github Homepage

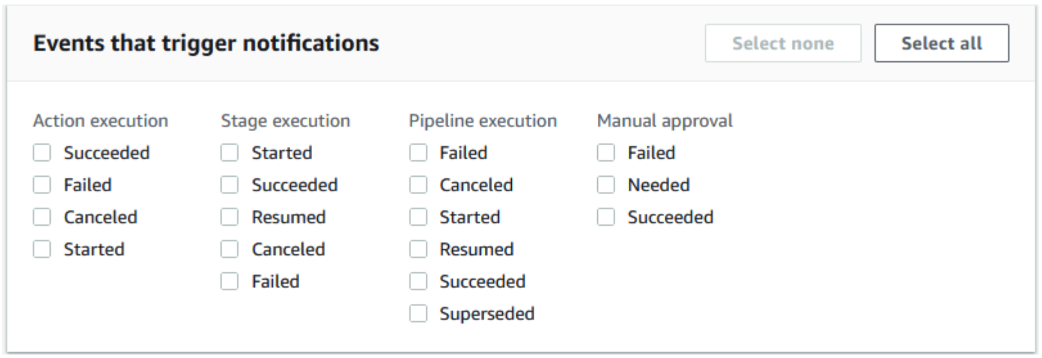
- Build
  CloudBees Homepage
  Jenkins Homepage
  TeamCity Homepage

- Test
  BlazeMeter Homepage
  Ghost Inspector Homepage
  Runscope Homepage

Custom actions can always be triggered from the CodeBuild console. For example SonarCloud can be called with: *sonar-scanner -Dsonar.organization=my_organization -Dsonar.projectKey=my_project -Dsonar.sources=src*.
Actions can be added to a CodePipeline by adding an additional step/action under action provider inside an action.

2.8 Monitoring / Notifications

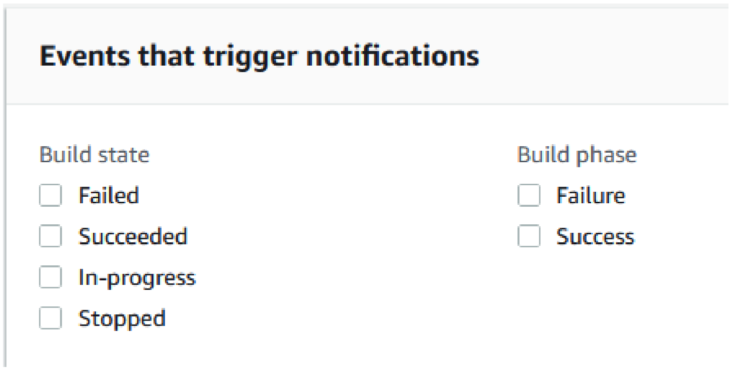
Each used service allows custom notifications and monitoring metrics. For example, CodePipeline allows the following event triggers:


Events that trigger notifications

Action execution	Stage execution	Pipeline execution	Manual approval
☐ Succeeded	☐ Started	☐ Failed	☐ Failed
☐ Failed	☐ Succeeded	☐ Canceled	☐ Needed
☐ Canceled	☐ Resumed	☐ Started	☐ Succeeded
☐ Started	☐ Canceled	☐ Resumed	
	☐ Failed	☐ Succeeded	
		☐ Superseded	

Figure 6: CodePipeline event notification triggers

Each notification rule can target a specific SNS role or an AWS Chatbot (Slack). An Amazon SNS topic is a logical access point that acts as a communication channel. A topic allows grouping multiple endpoints (such as AWS Lambda, Amazon SQS, HTTP/S, or an email address). SNS topics can be used as a messaging service for both applications and personal communication. An individual could subscribe to a certain topic via email to receive notifications about the pipeline status.
CodeBuild has similar trigger events:


Events that trigger notifications

Build state	Build phase
☐ Failed	☐ Failure
☐ Succeeded	☐ Success
☐ In-progress	
☐ Stopped	

Figure 7: CloudBuild notification trigger events

Elastic Beanstalk is entirely different. Here an email address can be used to receive important events from a specific environment. Important events can be defined as health checks for the application or as a monitoring alarm. Alarms can be added by choosing different metrics when creating the environment. Each environment is separate from each other, so configurations for each environment must be done by hand or by loading a pre-configured configuration.

2.9 Pricing

AWS provides a comprehensive calculator to calculate expenses: AWS Pricing Calculator. The next subsections will cover pricing models for each service used in the CI / CD pipeline.

2.9.1 CodeBuild

The AWS CodeBuild free tier includes 100 build minutes of build.general1.small per month. The CodeBuild free tier does not expire automatically at the end of your 12-month AWS Free Tier term and is used in this project.
If more resources are needed to build projects, other instances can be provisioned as well:

Compute instance type	Memory	vCPU	Linux price per build minute	Windows price per build minute
general1.small	3 GB	2	$0.005	N/A
general1.medium	7 GB	4	$0.01	N/A
arm1.large	16 GiB	8	$0.0175	N/A
general1.large	15 GB	8	$0.02	N/A
general1.2xlarge	144 GiB	72	$0.25	N/A
gpu1.large	244 GiB	32	$0.80	N/A

Table 1: CodeBuild compute instance pricing

Additional charges may be applied if the builds transfer data or use other AWS services. For example, charges can come from Amazon CloudWatch Logs for build log streams, Amazon S3 for build artefact storage, and AWS Key Management Service for encryption. You may also incur additional charges if you use AWS CodeBuild with AWS CodePipeline.

Secret manager prices:

- $0.40 per secret per month
- $0.05 per 10,000 API calls

The parameter store from AWS offers an alternative but is a slimmed-down version of the secret store.

2.9.2 CodePipeline

Each active pipeline costs $1/month, although the first 30 days are free.

2.9.3 Container Registry

AWS offers new ECR customers 500 MB-month of storage for one year for your private repositories. Additionally, Amazon ECR offers a 50 GB-month of always-free storage for public repositories for a new or existing customer. 500 GB of data can be transferred to the internet for free from a public repository each month anonymously (without using an AWS account).
When signed up, or authenticated, 5 TB of data can be transferred to the internet from a public repository each month, with unlimited bandwidth for free when transferring data from a public repository to AWS compute resources in any AWS Region. The storage, however, is not free: Storage is $0.10 per GB-month for data stored in private or public repositories.

Data transfer "in" and "out" refers to transfer into and out of Amazon Elastic Container Registry. Data transferred between Amazon Elastic Container Registry and Amazon EC2 within a single region is free of charge (i.e., $0.00 per GB). Data transferred between Amazon Elastic Container Registry and Amazon EC2 in different regions will be charged at Internet Data Transfer rates on both sides of the transfer.

Data transferred from private repositories:

	Pricing per GB
Data Transfer IN	
All data transfer in	$0.00
Data Transfer OUT	
Up to 1 GB / Month	$0.00
Next 9.999 TB / Month	$0.09
Next 40 TB / Month	$0.08
Next 100 TB / Month	$0.07
Greater than 150 TB / Month	$0.05

Table 2: Private repository data transfer pricing

Prices for public repositories: For transfers greater than 5 TB / Month to non AWS Regions is $0.09 per GB

2.9.4 RDS

For the Amazon Relational Database Service, a few different options are possible; each option's price varies significantly. For this project, a DB.t3.micro was used, which can be seen in the tables below.

Multi AZ:
Select Create Replica in Different Zone to have Amazon RDS maintain a synchronous standby replica in a different Availability Zone than the DB instance. Amazon RDS will automatically failover to the standby in the case of a planned or unplanned outage of the primary.

On-Demand DB Instances:
On-Demand DB Instances let you pay for computing capacity by the hour your DB Instance runs with no long-term commitments. This reduces the costs and complexities of planning, purchasing, and maintaining hardware and transforms what are commonly high fixed costs into much smaller variable costs.

Reserved instances:
Amazon RDS Reserved Instances give the option to reserve a DB instance for a one or three year term and in turn receive a significant discount compared to the On-Demand Instance pricing for the DB instance. Amazon RDS provides three RI payment options: No Upfront, Partial Upfront, All Upfront.

Important to know: Reserved Instance prices don't cover storage or I/O costs.

Single AZ On-Demand (in between is omitted as there is too much):

Standard Instances - Current Generation	Price Per Hour
db.t3.micro	$0.021
db.t3.small	$0.042
db.t3.medium	$0.084
.....	.....
db.m5.16xlarge	$6.784
db.m5.24xlarge	$10.176

Table 3: Single AZ On-Demand

Single AZ Reserved Instance:

Payment Option	Upfront	Monthly	Effective Hourly	On-Demand Hourly
No Upfront	$0	$10.439	$0.014	$0.0210
Partial Upfront	$60	$4.964	$0.014	$0.0210
All Upfront	$117	$0.000	$0.013	$0.0210

Table 4: 1 Year db.t3.micro prices for Frankfurt

2.9.5 Elastic Beanstalk

Amazon Elastic Beanstalk has more complicated pricing. The Elastic Beanstalk service is free of charge. However, the underlying systems that are used to host are not.

Elastic Beanstalk uses two components in the background:

- Amazon EC2 instances
- and/or Amazon S3 buckets

There are different pricing models for EC2 instances:

- On-Demand
  With On-Demand instances, you pay for computing capacity by the hour or the second depending on which instances you run.
- Spot instances
  Amazon EC2 Spot instances allow you to request spare Amazon EC2 computing capacity for up to 90% off the On-Demand price.

- Savings Plans
  Savings Plans are a flexible pricing model that offer low prices on EC2 and Fargate usage, in exchange for a commitment to a consistent amount of usage (measured in $/hour) for a 1 or 3-year term.

- Reserved Instances
  Reserved Instances provide you with a significant discount (up to 75%) compared to On-Demand instance pricing.

For this project, only On-Demand and Spot instances were used for testing. For instances, t2.micro (1vCPUs, 1GiB) and t2.small (1vCPUs, 2GiB) were chosen.

Spot instance prices (in between prices omitted as there were too many):

General Purpose	**Linux/UNIX Usage**	**Windows Usage**
....		
t2.micro	\$0.004 per Hour	\$0.0086 per Hour
....		
t3.medium	\$0.0144 per Hour	\$0.0328 per Hour
....		
m5a.24xlarge	\$1.671 per Hour	\$6.087 per Hour

Table 5: EC2 Spot Instance pricing

EC2 On-Demand prices (in between prices omitted as there were too many):

General Purpose	**vCPU**	**Memory (GiB)**	**Linux/UNIX Usage**
t2.micro	1	1 GiB	\$0.0134 per Hour
....			
t3.medium	2	4 GiB	\$0.048 per Hour
...			
m5a.24xlarge	96	384 GiB	\$4.992 per Hour

Table 6: EC2 On-Demand Pricing

When comparing table 5 and table 6, we can clearly see that On-Demand is more expensive than Spot Instances.

For Amazon S3 bucket pricing, only the S3 Standard general-purpose storage for any type of data, typically used for frequently accessed data is used. However, multiple Tiers exists here as well.

	Storage pricing
S3 Standard	
First 50 TB / Month	\$0.0245 per GB
Next 450 TB / Month	\$0.0235 per GB
Over 500 TB / Month	\$0.0225 per GB

Table 7:

2.10 Creating a CI / CD pipeline

This section describes each necessary step to create a pipeline. The resulting pipeline will look like Figure 8.

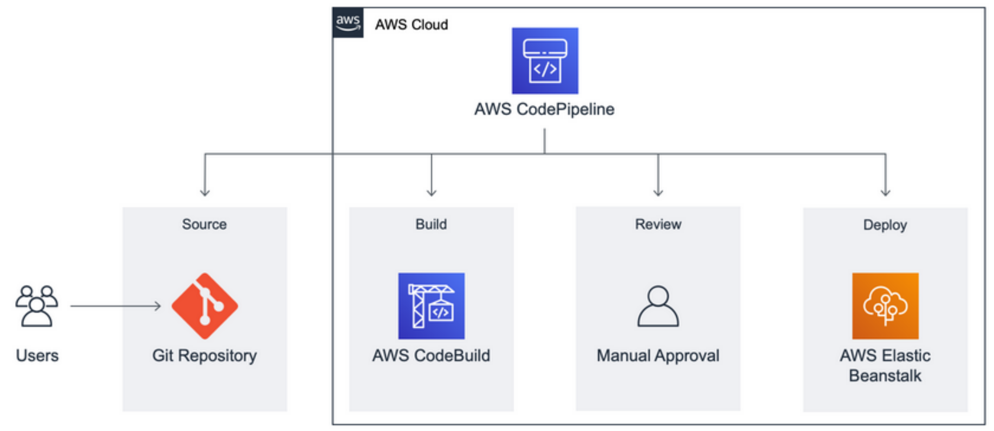


Figure 8: CI / CD Pipeline for the project

2.10.1 Setting up Elastic Container Registry

The AWS ECR does not automatically create entries for pushed container images; each container image needs a repository that has to be created beforehand. Each repository can have multiple versions of a single image. ECR only allows a push if a repository has already been created, resulting in a failure if attempted. The setup is simple; the repository needs a name and visibility settings. Advanced options include a scan on push and encryption, but this is not needed in most cases.

2.10.2 Setting up CodeBuild

Before you create a CodeBuild project, you need to review the IAM policies. It would help if you had CloudWatch Logs, S3, Systems Manager, CodeCommit, CodeBuild and Elastic Container Registry rights; otherwise, the build will fail.
Steps:

1. Go to AWS CodeBuild
2. Create a new build project
3. Select a project name
4. Select the VCS Repository (Github)
   If granted access, repositories should show up.
5. Add a webhook event
   Conditions for a webhook can be set, pushes to a branch can be monitored and tags are possible as well - insert at HEAD_REF: "̂refs/tags/" and the desired regex afterwards
   To allow pull request hooks, another filter group has to be added.
6. Manage the environment, Amazon Linux 2 is the recommended system as it provides packages and configurations with many AWS tools
7. Set the privileged flag as we want to build docker images
8. Additional specifications can be set for the build VM
   Timeouts, resources, certificates, file systems...
9. Add environment variables
   Available options are plaintext, parameter or the secrets manager
10. Set the build spec location; custom locations can be used as well

11. Artefacts produced by our build are the docker images and a "Dockerrun.aws.json" used in the CodePipeline to create instances.

12. Select CloudWatch logs, as they provide additional insight into how many builds failed and more important stats.

CodeBuild can be triggered manually, for a test run, this is recommended. An example of a cloudbuild.yml file:

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - ECR_REPOSITORY_URI_1="URI"
      - ECR_REPOSITORY_URI_2="URI"
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - AWS_IMAGE_TAG=build-$(echo $CODEBUILD_BUILD_ID | awk -F":" '{print $2}')
      - now=$(date)
      - docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
  build:
    commands:
      - echo Build started on $now with tag $AWS_IMAGE_TAG
      - echo Building the Docker image...
      - docker-compose -f docker-compose_aws.yml up -d --build
      - docker tag snippets_db:latest $ECR_REPOSITORY_URI_1:$AWS_IMAGE_TAG
      - docker tag snippets_db:latest $ECR_REPOSITORY_URI_1:latest
      - docker tag snippets_restapi:latest $ECR_REPOSITORY_URI_2:$AWS_IMAGE_TAG
      - docker tag snippets_restapi:latest $ECR_REPOSITORY_URI_2:latest
  post_build:
    commands:
      - echo Build completed on $now tag $AWS_IMAGE_TAG
      - echo Pushing the Docker images...
      - docker push $ECR_REPOSITORY_URI_1:$AWS_IMAGE_TAG
      - docker push $ECR_REPOSITORY_URI_1:latest
      - docker push $ECR_REPOSITORY_URI_2:$AWS_IMAGE_TAG
      - docker push $ECR_REPOSITORY_URI_2:latest
      - rm docker-compose.yml
artifacts:
  files:
    - 'Dockerrun.aws.json'
```

The "Dockerrun.aws.json" is important for the Elastic Beanstalk deployment. An example will be shown in later sections.

2.10.3 Setting up RDS

RDS setup has been simplified in recent years, there is an option "Easy create" which will create an RDS with the recommended best-practice configurations.

1. Open RDS and click "Create database"
2. Choose "Easy create"
3. Select PostgreSQL as engine type
4. Choose a DB instance size: Free tier is good for testing
5. Name DB instance identifier, which should be unique
6. Set a master username
7. Set a master password

More advanced options are also available:

- Connectivity
  Here the virtual private cloud (VPC) and security groups can be set. They allow/disallow access to RDS from different sources.
- Database authentication
  - Password authentication
    Authenticates using database passwords.

– Password and IAM database authentication
  Authenticates using the database password and user credentials through AWS IAM users and roles.

– Password and Kerberos authentication
  Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

As mentioned before, a VPC can be set as well, this important for the Elastic Beanstalk instance as it does not have access to RDS without setting the appropriate IAM permissions and security group. Each environment in Beanstalk that accesses an external RDS instance needs to be in the same security group as the RDS instance. When the setup is finished, the RDS system will be created, which can take a few minutes.

2.10.4 Setting up Elastic Beanstalk

Prerequisite:
Needs IAM permissions to load from ECR, error warnings give no info whatsoever if permissions are missing. Permissions can be set in the IAM console, a role and policy can be added with their respective buttons. Policies can either be added with a visual editor or by JSON.
Policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEbAuth",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowPull",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:ecr:*:YOUR_ID:repository/*"
            ],
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ]
        }
    ]
}
```

Normally, permission will be created when creating an environment. The permissions for ECR, however, are not included.
Steps:

1. Open Elastic Beanstalk
2. Create a new Application with a unique name
3. Create a new environment
   Choose between a Web server and Worker, the difference here is that Worker environments are meant for long-running workloads
4. Set the application name; it should be the same from before
5. Choose a name for the environment, it should be unique for an application and can not be changed later
6. Choose a platform where you application or code is run, for docker containers choose the Docker platform
7. Select either the sample application if no image has been pushed to the repository yet or upload your own code already
8. Click ”More options”
   This will open an advanced overview of the Elastic Beanstalk environment

9. Select "Single Instance" to try out the pipeline or select "High availability" to add a load balancer to the environment

10. Click on the "Edit" button under "Software" to add environment variables.
    Here environment variables can be set if they are not automatically saved when building the container

11. Under "Notifications" and "Monitoring" extra options can be set for health checks. If a load balancer has been selected, the paths to each service need to be specified as well.

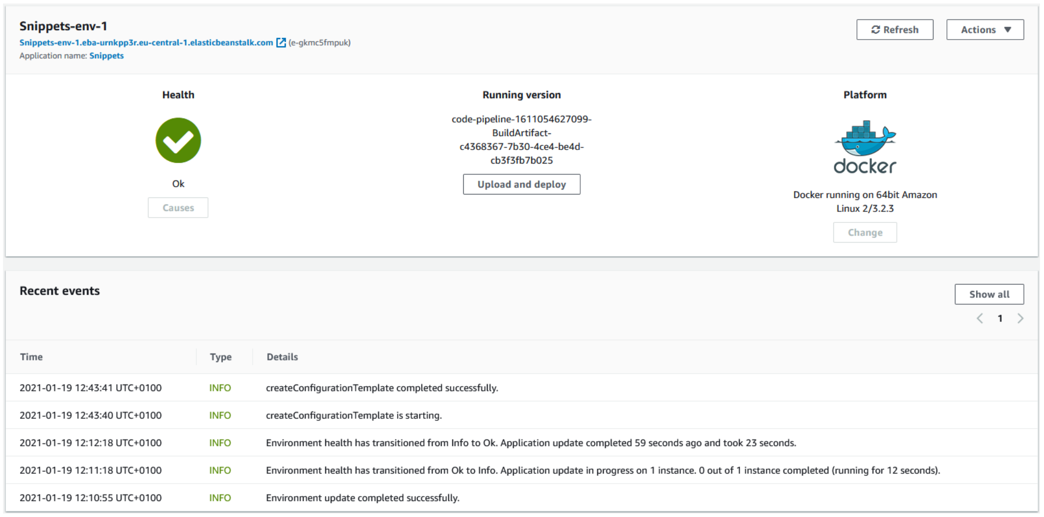
12. If an RDS instance should be coupled to the environment (for development), a database has to be specified. However, this database is not permanent and will be deleted if a new application version is added

13. Click "Create Environment"

Depending on what options have been set, initializing the environment can take a few minutes. An example for the "Dockerrun.aws.json" mentioned earlier:

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "436760934442.dkr.ecr.eu-central-1.amazonaws.com/snippets_restapi:late
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "80"
    }
  ]
}
```

This file allows the Elastic Beanstalk to add and start a specified application container. Due to how the CodePipeline works, this file is needed as output of the pipeline, otherwise starting a container is not possible. After the environment has been created, the following should be displayed:



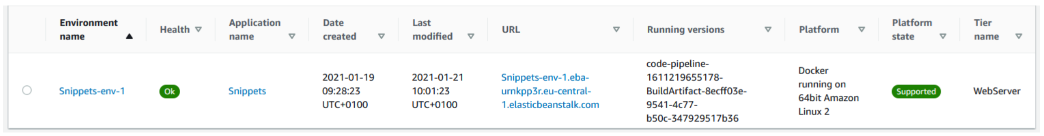
Snippets-env-1
Snippets-env-1.eba-urnkpp3r.eu-central-1.elasticbeanstalk.com (e-gkmc5fmpuk)
Application name: Snippets

Health	Running version	Platform
Ok	code-pipeline-1611054627099-BuildArtifact-c4368367-7b30-4ce4-be4d-cb3f3fb7b025	Docker running on 64bit Amazon Linux 2/3.2.3

Recent events

Time	Type	Details
2021-01-19 12:43:41 UTC+0100	INFO	createConfigurationTemplate completed successfully.
2021-01-19 12:43:40 UTC+0100	INFO	createConfigurationTemplate is starting.
2021-01-19 12:12:18 UTC+0100	INFO	Environment health has transitioned from Info to Ok. Application update completed 59 seconds ago and took 23 seconds.
2021-01-19 12:11:18 UTC+0100	INFO	Environment health has transitioned from Ok to Info. Application update in progress on 1 instance. 0 out of 1 instance completed (running for 12 seconds).
2021-01-19 12:10:55 UTC+0100	INFO	Environment update completed successfully.

Figure 9: Development environment



Environment name	Health	Application name	Date created	Last modified	URL	Running versions	Platform	Platform state	Tier name
Snippets-env-1	Ok	Snippets	2021-01-19 09:28:23 UTC+0100	2021-01-21 10:01:23 UTC+0100	Snippets-env-1.eba-urnkpp3r.eu-central-1.elasticbeanstalk.com	code-pipeline-1611219655178-BuildArtifact-8ecff03e-9541-4c77-b50c-347929517b36	Docker running on 64bit Amazon Linux 2	Supported	WebServer

Figure 10: Environment events

Multiple environments can be added; we choose to use one environment for development and one for staging.

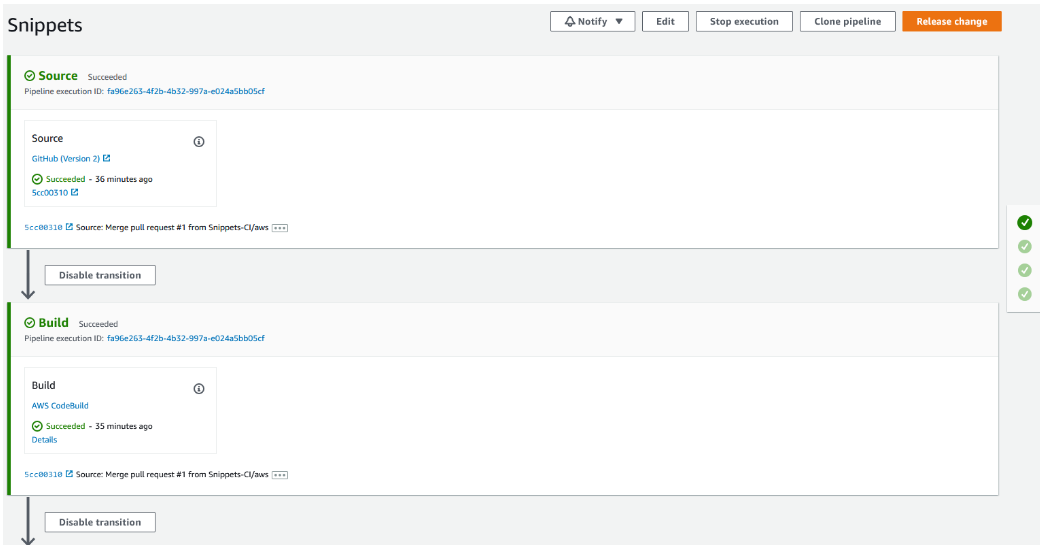
2.10.5 Creating the pipeline

Multiple Pipelines can be created, though usually one pipeline per application is created to avoid unnecessary confusion. The pipeline itself has steps to deploy to a different provider.
Steps:

- Open CodePipeline
- Choose a name
- Select "New service role". Although an existing one can be used, it is better to let AWS automatically create the needed role with policies to avoid IAM permission problems.

- Check ”Allow AWS CodePipeline to create a service role so it can be used with this new pipeline” otherwise permission problems will arise.
- Click ”Next”
- Choose a source provider, this can be a VCS or AWS specific source. It is also possible to react to pushes in the ECR (Elastic Container Registry) which allows skipping the build stage. For this project, we select Github as we want the build step to be a part of the pipeline.
- Click ”Next”
- Select AWS CodeBuild as build provider and choose the project name created in the CodeBuild step
- Click ”Next”
- Select a deploy provider, in this case, Elastic Beanstalk
- Select the application and environment created before
- Review the pipeline and create it which creates a basic pipeline
- Select the created pipeline and click ”Edit” to add additional steps, in our case we add a manual approval and additional deployment to another Elastic Beanstalk Environment
- Add a new stage and give it a name; it can not be changed later on
- Click ”Edit stage” to add a new action to another action group, for the manual approval the action ”Manual approval” has to be selected. Actions after this will only run if approval has been given
- Add another deploy action, this time for another Elastic Beanstalk Environment

The finished pipeline with a successful run should look like this:



Source Succeeded
Pipeline execution ID: fa96e263-4f2b-4b32-997a-e024a5bb05cf

Source
GitHub (Version 2)
Succeeded - 36 minutes ago
5cc00310

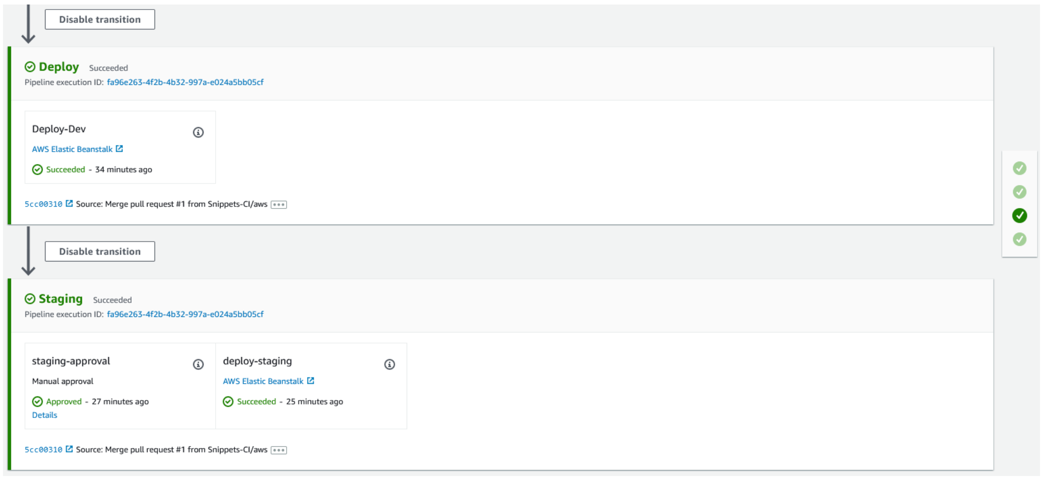
5cc00310 Source: Merge pull request #1 from Snippets-CI/aws

Build Succeeded
Pipeline execution ID: fa96e263-4f2b-4b32-997a-e024a5bb05cf

Build
AWS CodeBuild
Succeeded - 35 minutes ago
Details

5cc00310 Source: Merge pull request #1 from Snippets-CI/aws

Figure 11: Pipeline



Deploy Succeeded
Pipeline execution ID: fa96e263-4f2b-4b32-997a-e024a5bb05cf

Deploy-Dev
AWS Elastic Beanstalk
Succeeded - 34 minutes ago

5cc00310 Source: Merge pull request #1 from Snippets-CI/aws

Staging Succeeded
Pipeline execution ID: fa96e263-4f2b-4b32-997a-e024a5bb05cf

staging-approval
Manual approval
Approved - 27 minutes ago
Details

deploy-staging
AWS Elastic Beanstalk
Succeeded - 25 minutes ago

5cc00310 Source: Merge pull request #1 from Snippets-CI/aws

Figure 12: Pipeline

Now a push to the VCS will trigger the pipeline and automatically deploy a new version to Elastic Beanstalk.

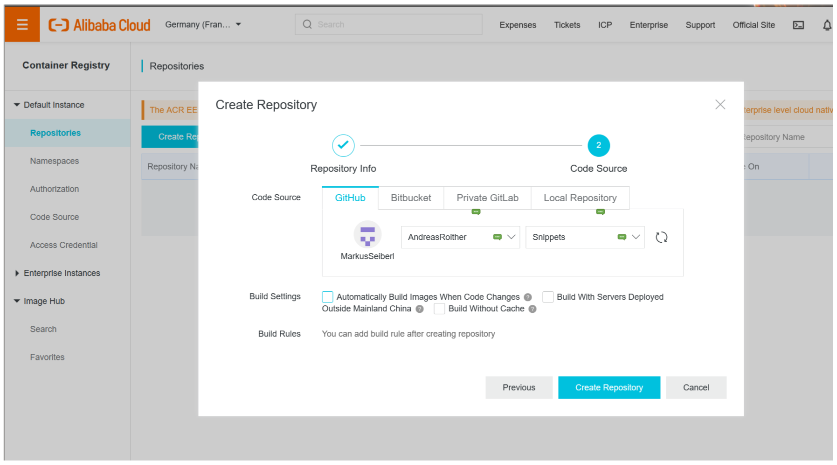
3 Alibaba

This section describes the CI / CD process for Alibaba Cloud.

3.1 Integration with different Version Control Systems

In contrast to other Cloud Provider, Alibaba Cloud does not offer a service for continuous integration. However, the Docker Registry can be connected with following Version Control Systems:

- GitHub
- Bitbucket
- Private GitLab
- Local Repository



Create Repository

Code Source: GitHub; AndreasRoither; Snippets

Build Settings: Automatically Build Images When Code Changes; Build With Servers Deployed Outside Mainland China; Build Without Cache

Build Rules: You can add build rule after creating repository

Figure 13: Different VCS integrations

3.2 Cloud Build

Alibaba Cloud only offers a cloud build of container images. Container Registry automates the building and delivery of images from source code repositories to Container Registry repositories based on Dockerfiles. When the source code is updated, the Container Registry automatically builds an image using Dockerfiles and uploading the image to Container Registry repositories. Docker-compose files are not supported. The build trigger needs to be defined multiple times for building multiple images, each with the corresponding Dockerfile.

A build trigger can either be branch- or tag-based. Regular expressions are supported.

3.3 Pipelines

A pipeline needs to be setup manually on an Elastic Compute Service (ECS) instance as Alibaba Cloud currently does not offer a "Pipeline"-Service for Continuous Integration.

3.4 Deployment Strategies

The Container Service for Kubernetes supports different deployment strategies:

- Rolling Update
- Blue/Green Deployment (manually create new Deployment and change services to point to the new deployment)
- Canary Deployment

3.5 Environments

The Alibaba Cloud service API Gateway allows configuring the test, staging, and production environments for an API operation, which correspond to the Test, Pre, and Release environments in the API Gateway console, respectively.

The clients can bind the API group, to which the API operation belongs, to different domain names for different environments. Alternatively, they can add the **X-Ca-Stage** parameter as a request parameter of the API operation. The use of environment variables is also possible.

3.6 Scaling

For the Container Service for Kubernetes Alibaba Cloud offers the Auto Scaling (ESS) service, which can dynamically scale computing resources. ESS defines elasticity from the following aspects:

- Workload scaling. ESS can adjust workloads, such as pods. For example, Horizontal Pod Autoscaler (HPA) is a typical workload scaling component that can change the number of pod replicas to scale the workload.

- Resource scaling. If the resources of a cluster cannot meet the requirements of the scaled workload, the related component automatically adds Elastic Compute Service (ECS) instances or elastic container instances (ECIs) to the cluster.

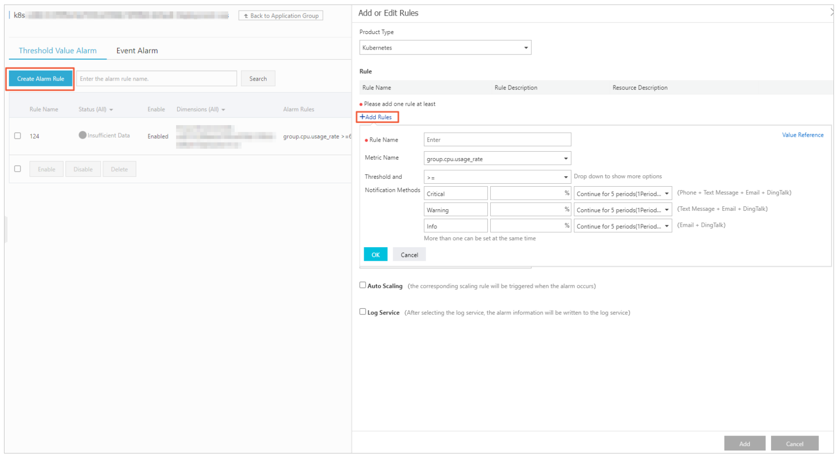
The CPU and memory resources of a node are allocated based on requests from pods. When pods on a node are overwhelmed by resource requests, the pods become pending. The auto-scaling component then automatically calculates the number of nodes required to be added based on the scaling group's predefined resource specifications and constraints. If a node belongs to a scaling group and the requested resources of the pods on the node are less than the threshold, the auto-scaling component automatically removes the node from the cluster.

3.7 Extensions

Alibaba Cloud Marketplace is an online platform for global independent software vendors (ISVs) to sell their products and services to customers.
The difference to other cloud providers is that the current software stack's functionality is not extended, rather they provide custom solutions with the services Alibaba Cloud already offers.

3.8 Monitoring / Notifications

The Container Service for Kubernetes includes a resource monitoring which allows to view resource usage of workloads. The resources include CPU, memory, and network resources. ACK integrates Cloud Monitor to monitor resources. By default, ACK installs Cloud Monitor for all new clusters.



Rule Name	Status (All)	Enable	Dimensions (All)	Alarm Rules
124	Insufficient Data	Enabled	[redacted]	group.cpu.usage_rate >=6

Product Type: Kubernetes

Rule

Rule Name	Rule Description	Resource Description

Please add one rule at least

Rule Name: Enter

Metric Name: group.cpu.usage_rate

Threshold and Notification Methods: >= Drop down to show more options

Critical	%	Continue for 5 periods(1Period...	(Phone + Text Message + Email + DingTalk)
Warning	%	Continue for 5 periods(1Period...	(Text Message + Email + DingTalk)
Info	%	Continue for 5 periods(1Period...	(Email + DingTalk)

More than one can be set at the same time

Auto Scaling (the corresponding scaling rule will be triggered when the alarm occurs)

Log Service (After selecting the log service, the alarm information will be written to the log service)

Figure 14: Resource Monitoring - Rule creation

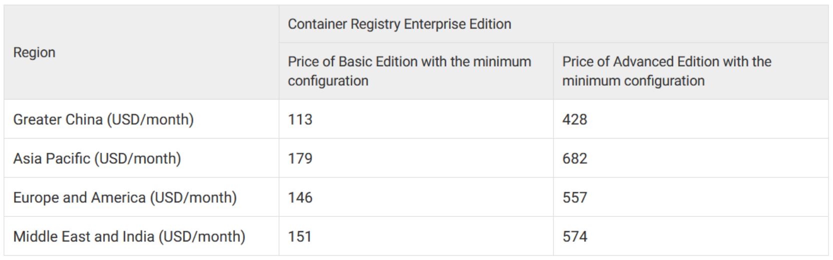
Additionally to resource monitoring of the Container Service for Kubernetes, Prometheus, an open-source tool, can be deployed within the Kubernetes cluster.

3.9 Pricing

Alibaba Cloud provides a comprehensive calculator to calculate expenses: Price Calculator.

3.9.1 Container Registry

The Container Registry Default Instance Edition is free of charge. The costs for the Enterprise Edition can be depicted in Figure 15.



Region	Container Registry Enterprise Edition	
	Price of Basic Edition with the minimum configuration	Price of Advanced Edition with the minimum configuration
Greater China (USD/month)	113	428
Asia Pacific (USD/month)	179	682
Europe and America (USD/month)	146	557
Middle East and India (USD/month)	151	574

Figure 15: Pricing of Container Registry Enterprise Edition

3.9.2 Elastic Compute Service

For Elastic Compute Service (ECS) Alibaba Cloud offers 3 options of payment:

- Saving Plans
- Reserved Instances
- Pay-As-You-Go
- Subscription

For example, an instance with Linux as the operating system starts at \$0.007 per hour with 1 vCPU core and 0.5 GB of memory.

3.9.3 Server Load Balancer

Fees are calculated based on the length of time of load balancer rentals and network traffic. For the EU Central Region, the fee is \$0.006 per hour and \$0.07 per GB.

3.9.4 Elastic IP

The pay-by-data-transfer option for Elastic IP costs the Region Germany \$0.006 and \$0.07 for every GB of data transfer.

3.10 Creating a CI / CD pipeline

It is important to note that Jenkins with Kubernetes is used to create the CI / CD pipeline.

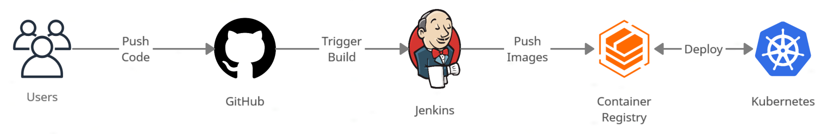


Figure 16: CI / CD Pipeline

Basic steps include:

1. Creating the Container Registry
2. Creating the PostgreSQL
3. Creating an ECS instance
4. Setting up Jenkins
5. Creating Kubernetes incl. triggers

3.10.1 Creating the Container Registry

1. Navigate to the Container Registry in the Alibaba Cloud Console.
2. Click on "Namespace"
3. Create a new Namespace
4. Switch to "Repositories"
5. Click on "Create Repositry"
6. Select the Region, Namespace and repository type
7. Enter a meaningful repository name and a summary
8. Click "Next"
9. Select "Local Repository"
10. Click on "Create Repository"

3.10.2 Creating the PostgreSQL

1. Navigate to the ApsaraDB for RDS in the Alibaba Cloud Console.
2. Click on "Create Instance"
   (a) Select a appropriate Region
   (b) Select PostreSQL 13 as Database Engine
   (c) Select an for the application appropriate instance type
3. Click on "Next: Instance Configuration"

4. Click "Next: Confirm Order"
5. Click "Pay Now"

3.10.3 Creating an ECS instance

1. Navigate to the Elastic Compute Service in the Alibaba Cloud Console.
2. Click on "Create ECS Instance"
   (a) Select a appropriate region
   (b) Select a appropriate instance type
   (c) Select "Ubuntu 20.04 64-bit" as Image
3. Click on "Next: Networking"
   (a) Enable Port 80 and 443
4. Click on "Next: System Configurations"
   (a) Select "Password" as logon credentials and enter a secure password
   (b) Enter a meaningful hostname
5. Click on "Preview"
6. Click on "Create Instance"

3.10.4 Setting up Jenkins

In order to install Jenkins, console access is needed. Alibaba Cloud offers a web-based cloud shell. However the Keyboard layout is by default the US-International. Therefore, we recommend to connect with e.g. Putty or Terminal via SSH to the ECS instance. The IP address of the ECS instance can be found in the Alibaba Cloud Console in the section Elastic Compute Service.
Execute these commands:

1. wget https://labex-ali-data.oss-us-west-1.aliyuncs.com/spark-analysis/jdk-8u181-linux-x64.tar.gz
2. tar -zxf jdk-8u181-linux-x64.tar.gz -C /usr/local

3. nano /etc/profile
   - # Add these two lines at the end:
   - export JAVA_HOME=/usr/local/jdk1.8.0_181
   - export PATH=$PATH:$JAVA_HOME/bin
4. source /etc/profile
5. wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key — sudo apt-key add -
6. echo "deb https://pkg.jenkins.io/debian/stable binary/" — sudo tee /etc/apt/sources.list.d/jenkins.list
7. apt update
8. apt -y install jenkins git
9. nano /etc/init.d/jenkins
   - # Append the JDK path to the PATH variable
   - :/usr/local/jdk1.8.0-181/bin
10. systemctl daemon-reload
11. systemctl start jenkins
12. systemctl status jenkins
13. apt -y install nginx
14. nano /etc/nginx/sites-enabled/default
   - # Add the three lines at the beginning of the file
   - upstream jenkins
   - server 127.0.0.1:8080;
15. systemctl start nginx

As the current Security Group of the ECS instance blocks all incoming traffic on the port 8080, we need to whitelist the port. Go to a browser window and input the following: ”http://IP_ADDRESS:8080”.

1. Follow the onscreen instructions to set up Jenkins.
2. The initial password can be found by typing in the command
   - cat /var/lib/Jenkins/secrets/initialAdminPassword
3. Install the suggested plugins.
4. Again follow the instructions.

For setting up a build pipeline we use the Freestyle project:

1. Click on ”Create new jobs”
2. Select ”Freestyle project” and enter a meaningful name
3. Go to ”Source Code Management”
4. Select Git and enter the GitHub repository URL
5. By clicking ”Add” on the credentials add the login credentials for the GitHub repository.
6. Go to the ”Build” section and add the build step ”Execute shell”
   - Add following commands:

   -
     ```
     CONTAINER_REGISTRY_USER='CR_USER_NAME'
     CONTAINER_REGISTRY_PASSWD='SECURE_PASSWORD'
     CONTAINER_REGISTRY_URL='URL_TO_CR'
     CONTAINER_REGISTRY_REPO='MY_REPO'

     sudo docker login
        -u $(CONTAINER_REGISTRY_USER)
        -p $(CONTAINER_REGISTRY_PASSWD)
        $(CONTAINER_REGISTRY_URL)

     sudo docker build
     ```

```
            -t $(CONTAINER_REGISTRY_URL)/
            $(CONTAINER_REGISTRY_REPO):$(BUILD_TAG) .

        sudo docker push
            $(CONTAINER_REGISTRY_URL)/
            $(CONTAINER_REGISTRY_REPO):$(BUILD_TAG)
```

7. Click on "Save"

8. Start the build.

3.10.5 Creating Kubernetes Cluster

We use the Serverless Kubernetes option for demonstration purposes, as it is the fastest way to setup.

1. Navigate to "Container Service for Kubernetes" in the Alibaba Cloud Console.

2. Click on "Create Kubernetes Cluster"

3. Select the "Serverless Kubernetes"

    - Select a appropriate region
    - Select a valid VPC

4. Click on "Create Cluster"

5. In order to add environment variables click on the created cluster

6. Navigate to "Deployments"

7. Click on "Create from Image"

8. Add a name for the deployment and select the number of replicas

9. Click "Next"

10. Select the Image

11. Add specific ports (regarding the container)

12. Add possible environment variables (e.g. database access credentials)

13. Click ”Next”
14. Click ”Create”
15. Click ”View Details”
16. Navigate to ”Triggers”
17. Click on ”Create Trigger”
18. Confirm and copy the URL
19. Navigate to the Container Registry
20. Click on ”Manage” on the previously created repository
21. Navigate to ”Trigger”
22. Click on ”Create”
23. Enter a meaningful name and the trigger URL
24. Click ”OK”

4 Azure

This section describes the CI / CD process for Azure.

Azure DevOps is a Software as a service (SaaS) platform from Microsoft that provides an end-to-end DevOps toolchain for developing and deploying software. It also integrates with most leading tools on the market and is a great option for orchestrating a DevOps toolchain.

Azure DevOps comprises a range of services covering the full development life-cycle:

- Azure Boards: agile planning, work item tracking, reporting and visualisation tool.
- Azure Pipelines: a cloud agnostic CI/CD platform with support for containers or Kubernetes.
- Azure Repos: a cloud-hosted git repository.
- Azure Test Plans: provides a integrated planned and exploratory testing solution.
- Azure Artifacts: provides integrated package management with support for Maven, npm, Python and NuGet package feeds.

The integrated Marketplace enables customisation and flexibility.

4.1 Integration with different Version Control Systems

Git is the default version control provider for new projects. As an alternative option, Team Foundation Version Control (TFVC) can be selected. An import to Azure DevOps is possible from every git repository. Only requirement is the URL for cloning. If the repository requires authorization username and password can be provided.

Although cloning a git repository into Azure Repos is possible, it is not a hard requirement for building a CI/CD pipeline.

4.2 Cloud Build

Azure DevOps offers services to build and test the code. The pipeline can be changed either via GUI provided by Azure DevOps or by editing the yaml file.

The yaml file defines the triggers, platforms, jobs and their steps. Existing functionality can be extended by downloading extensions from the Marketplace.

4.3 Pipelines

The Azure Pipelines supports automatically build and test of code projects to make them available to others. It works with just about any language or project type. Azure Pipelines supports continuous integration (CI) and continuous delivery (CD) to constantly and consistently test and build code and ship it to any target.

Azure DevOps distinguishes between two Pipelines:

- Build Pipelines and
- Release Pipelines.

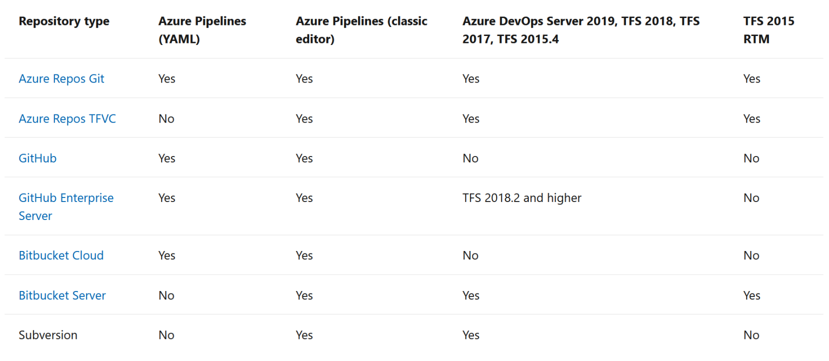
Although Build Pipelines have the ability to deploy code or containers, Release Pipelines offer more features for customizing the releases.

Triggers for example can be:

- Scheduled triggers: configure a pipeline to run on a schedule.
- CI triggers: causes the pipeline to run whenever a specified tag or branch is pushed.
- PR triggers: causes to run whenever a pull request is opened with one of the specified target branches, or when updates are made to such a pull request.
- Comment triggers: repository collaborators can comment on a pull request to manually run a pipeline.

4.3.1 Build Pipeline

Figure 17 depicts the currently supported source repositories for Azure DevOps Pipelines.



Repository type	Azure Pipelines (YAML)	Azure Pipelines (classic editor)	Azure DevOps Server 2019, TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Azure Repos Git	Yes	Yes	Yes	Yes
Azure Repos TFVC	No	Yes	Yes	Yes
GitHub	Yes	Yes	No	No
GitHub Enterprise Server	Yes	Yes	TFS 2018.2 and higher	No
Bitbucket Cloud	Yes	Yes	No	No
Bitbucket Server	No	Yes	Yes	Yes
Subversion	No	Yes	Yes	No

Figure 17: Supported source repositories

Via the yaml file triggers can be defined. Depending on the source repository different triggers are available.

4.3.2 Release Pipeline

Release pipelines in Azure Pipelines helps teams continuously deliver software to customers at a faster pace and with lower risk. It offers fully automated testing and delivery of your software in multiple stages all the way to production. Or, set up semi-automated processes with approvals and on-demand deployments.

To author a release pipeline, one must specify the artifacts that make up the application and the release pipeline. An artifact is a deployable component of your application. It's typically produced through a Continuous Integration or a build pipeline. Azure Pipelines releases can deploy artifacts produced by a wide range of artifact sources. such as Azure Pipelines build, Jenkins, or Team City. The release pipeline is defined by stages and restricts deployments into or out of a stage using approvals. Define the automation in each stage using jobs and tasks. Release Pipelines allow the use of variables to generalize the automation. Figure 18 shows a possible way to implement a release pipeline with two different environments.

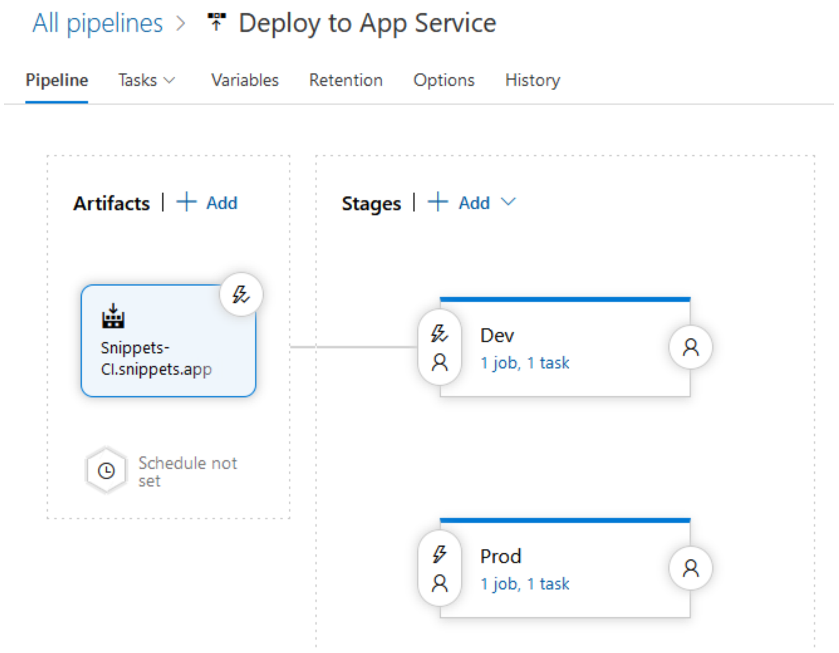


Figure 18: Example of a release pipeline

4.4 Deployment Strategies

Out-of-the-box deployment strategies vary greatly between different deployment targets. While the Kubernetes manifest task currently only supports canary deployment strategy, Virtual Machines support:

- Blue-Green Deployment,
- Canary Deployment and
- Rolling Deployment.

The Azure App Service support traffic splitting between different slots.

4.5 Environments

An environment is a collection of resources, such as Kubernetes clusters and virtual machines, that can be targeted by deployments from a pipeline.

4.6 Scaling

4.6.1 Azure App Service

Azure App Service supports vertical as well as horizontal scaling. For scale out auto-scaling is available. Rules define not only a minimum and maximum of instances which can be set but also criterias. Criterias determine if a certain metric like the CPU percentage is below or above a certain threshold. Furthermore, time-based scaling can also be enabled, where instances can be up- or down-scaled depending on the time.

4.6.2 Azure Kubernetes Service

The cluster autoscaler component can watch for pods in your cluster that can't be scheduled because of resource constraints. When issues are detected, the number of nodes in a node pool is increased to meet the application demand. Nodes are also regularly checked for a lack of running pods, with the number of nodes then decreased as needed.

AKS clusters can scale in one of two ways:

- The cluster autoscaler watches for pods that can't be scheduled on nodes because of resource constraints. The cluster then automatically increases the number of nodes.
- The horizontal pod autoscaler uses the Metrics Server in a Kubernetes cluster to monitor the resource demand of pods. If an application needs more resources, the number of pods is automatically increased to meet the demand.

4.7 Extensions

Extensions are add-ons you can use to customize and extend your DevOps experience with Azure DevOps. Some extensions provide new Azure Pipleines taks that teams can use in their builds. Examples of extensions:

- SonarQube
- Replace Tokens
- Kubernets extension

- Release Management Utility tasks
- Google Play
- Apple App Store

4.8 Monitoring / Notifications

Azure DevOps allows to configure the notification settings. These settings are limited to email notifications. However extensions for Slack or MS Teams are available.

- Build
  - Build Completes
  - Build fails
- Pipelines
  - Run stage waiting for approval
  - Run stage waiting for Manual validation
- Release
  - An approval for a deployment is pending
  - A deployment is completed
  - A request for release creation failed
  - A manual intervention for a deployment is pending

4.9 Pricing

For Azure products, Azure provides a comprehensive pricing calculator: Azure Pricing Calculator.

4.9.1 Azure DevOps

Azure offers two different plans regarding Azure DevOps. Clients can choose between Individual Services like Azure Pipelines or Azure Artifacts and User Licenses which include several services.

The first 5 Users for the **Basic Plan** are free of cost. For each additional user a monthly cost of 6$ gets added. The plan includes:

- Azure Pipelines,
- Azure Boards,
- Azure Repos and
- Azure Artifacts.

Azure Pipelines as Individual Service includes 1 Free Microsoft-hosted CI/CD job with a limited time of 1800 minutes per month. The costs increases by $40 per month for each additional parallel job.

4.9.2 Azure Database for PostgreSQL

Azure offers a Single Server, a Flexible Server and a Hyperscale. Clients can choose between Basic, Gneral Purpose and Memory Optimized pricing tiers. A detailed overview about the prices can be here.

4.9.3 Container Registry

	Basic	**Standard**	**Premium**
Price per day	$0.167	$0.667	$1.667
Included storage (GB)	10	100	500
Total web hooks	2	10	500
Geo Replication	Not Supported	Not Supported	Supported $1.667 per replicated region

Additional storage is available at a daily rate for all service tiers at a price of $0.003/day/GB.

4.9.4 Azure App Service

The price for the Azure App depends on the selected Plan (Basic, Standard, Premium, Isolated). In each plan the different instance types can be selected all varying in price.

4.10 Creating a CI / CD pipeline

This section describes the necessary steps for creating a basic CI / CD pipeline. Overall the job can be broken down into five small tasks:

1. Creating Azure Container Registry
2. Creating Azure Database for PostgreSQL
3. Creating Azure App Service
4. Creating a Build Pipeline in Azure DevOps
5. Creating a Release Pipeline in Azure DevOps

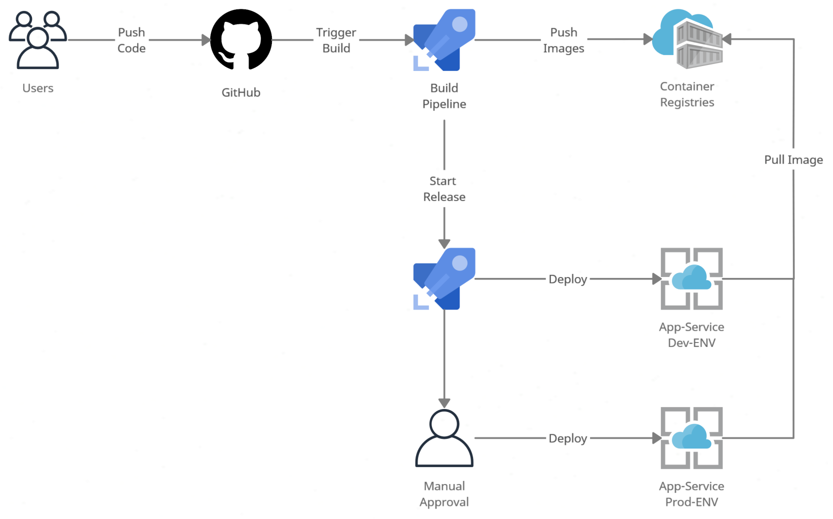


Figure 19: CI / CD Pipeline

4.10.1 Creating Azure Container Registry

1. Go to Azure Portal
2. Select the desired Resource Group (create one if needed)
3. Click "Add"
4. Select "Container Registry"
5. Enter a Resource group, enter a valid (and) unique name, select the location and a SKU.
6. Press "Create + Review"
7. Press "Create" (This may take a while)
8. After the deployment finished, press on the "Go to resource" button.

Before we can jump to the next task, we need a way to access the registry. Therefore,

1. go to "Settings" → "Access Keys"
2. Enable the Admin user

4.10.2 Creating Azure Database for PostgreSQL

1. Go to the desired Resource Group
2. Click "Add"
3. Select "Azure Database for PostgreSQL"
4. Click on "Create" at the desired option. We will use a Single server as it is perfectly adequate for a basic application.
5. Select or enter values for:
   (a) Resource group
   (b) Server name
   (c) Data Source (Select the Backup option if needed)

   (d) Location
   (e) Version (of PostgreSQL)
   (f) Compute + Storage
   (g) Admin username
   (h) Admin password

6. Click on "Review + Create"

7. Click on "Create"

4.10.3 Creating Azure App Service

1. Go to the desired Resource Group

2. Click "Add"

3. Select "Web App for Containers"

4. Select or enter values for:

   (a) Resource group
   (b) Name (needs to be unique)
   (c) Publish: Docker Container
   (d) Region
   (e) SKU and size (For staging slots a "Production" plan is needed)

5. Click on "Review + Create"

6. Click on "Create"

7. After the deployment finished, press on the "Go to resource" button.

Now we need to set the credentials for the Azure Container Registry and possible Environment Variables for the Docker container. Therefore:

1. Click on "Configuration"

2. Enter the values seen in section 4.10.1
   DOCKER_REGISTRY_SERVER_URL,
   DOCKER_REGISTRY_SERVER_USERNAME,
   DOCKER_REGISTRY_SERVER_PASSWORD
3. Add additional values as environment variables for the Docker container
4. Click "Save"

The next step, is to create an additional deployment slot:

1. Click on "Deployment slots"
2. Click "Add Slot"
3. Enter a suitable name and clone the settings
4. By selecting this slot and navigating to "Configuration", the environment variables can be changed for only this slot.

4.10.4 Creating a Build Pipeline in Azure DevOps

1. Go to Azure DevOps
2. Create a Project
3. Got to Pipelines → Create Pipeline
4. Select the Source Repository
5. Select the the "Docker Compose" and edit it by clicking on the "Settings" button above the task definition
   (a) Select As Container Registry Type "Azure Container Registry"
   (b) Select your Azure Subscribtion
   (c) Select the Azure Container Registry which we created in section 4.10.1
   (d) Enter the path to the docker-compoye.yml file
   (e) As Action select the "Build service images"
   (f) Optionally you can add additional image tags
6. Copy Paste this task and change the Action to "Push service images"
7. Click "Save"

4.10.5 Creating a Release Pipeline in Azure DevOps

1. Go to Pipelines → Releases
2. Click on ”New” → ”New release pipeline”
3. Select ”Empty Job”
4. Enter a suiting name like ”Dev” or ”Development”
5. Click ”Add” in the Artifacts section
6. Select ”Build” as source type
7. Select the previously created build pipeline
8. Click ”Add”
9. To create a trigger, click on the ”Continuous deployment trigger” symbol
10. Enable the ”Continuous deployment trigger” (this triggers the release pipeline every time a build pipeline finishes successfully)
11. Edit the stage by clicking on it
12. Click ”Add” next to the agent
13. Select the ”Azure App Service deploy” template
    (a) Enter a meaningful display name
    (b) Select the ”Azure Resource Manager” as Connection type
    (c) Select the Azure subscription
    (d) Select ”Web App for Containers (Linux)” as App Service type
    (e) Select the App Service which we created previously
    (f) Check the ”Deploy to Slot or App Service Environment”
    (g) Select a Slot
    (h) Enter the URL for the Azure Container Registry
    (i) Enter the name of the image

   (j) Enter the tag of the image

   (k) Click "Save"

14. Switch back to the "Pipeline" tab and add a new stage

15. Add a new "Azure App Service deploy" template to the new stage

16. Repeat the steps of the previous "Azure App Service deploy" template configuration with the only difference of selecting a different "Slot"

17. As Pre-deployment conditions various types of triggers can be selected.

18. Click "Save"

5 Google

This section describes the CI / CD process for Google.

5.1 Integration with different Version Control Systems

There are currently 3 supported version control systems:

- GCloud
- Github
- Bitbucket (only mirrored)

5.2 Cloud Build

Similar to AWS, Google also offers two services that can be used to create and upload containers: CloudBuild and the Container Registry. The important difference is that Google allows a push to the Container Registry without an explicit repository for the container image and is likely to save time if multiple images are created.

Since Gitlab was not mentioned above, as it is not supported yet. It is possible, however, to run GCloud build from scripts. So it is possible to use Gitlab given a valid Gitlab CI configuration file. Example of a Gitlab CI file:

```yaml
image: docker:latest

stages:
  - deploy

deploy:
  stage: deploy
  image: google/cloud-sdk
  services:
    - docker:dind
  script:
    - echo $GCP_SERVICE_KEY > gcloud-service-key.json
    - gcloud auth activate-service-account --key-file gcloud-service-key.json
    - gcloud config set project $GCP_PROJECT_ID
    - gcloud builds submit . --config=cloudbuild.yaml
```

The complete setup of a CloudBuild trigger and project will be explained later.

5.3 Pipelines

A simplified pipeline can be created with Cloud Run which is can be used to develop and deploy highly scalable containerized applications on a fully managed serverless platform. There is currently no solution provided by Google that is the exact same to CodePipeline from AWS. For this reason, we have to use a custom solution. Google offers a sample app that can be used for project setup. A Kubernetes cluster with GKE is used to install Jenkins with Helm. Jenkins is then used to set up a pipeline.

5.4 Deployment Strategies

Since we are using Kubernetes to deploy our app, a few different deployment strategies are available:

1. recreate:
   terminate the old version and release the new one

2. ramped:
   release a new version on a rolling update fashion, one after the other

3. blue/green:
   release a new version alongside the old version then switch traffic

4. canary:
   release a new version to a subset of users, then proceed to a full rollout

5. a/b testing:
   release a new version to a subset of users in a precise way

Google Cloud Run also features a few different strategies:

1. rollbacks

2. gradual roll-outs (blue-green deployment)

3. traffic migration (split traffic between two or more revisions)

5.5 Environments

Different environments can be created by using multiple Kubernetes clusters. Another option is to use namespaces for development or staging environments.

5.6 Scaling

Googles GKE cluster autoscaler automatically resizes the number of nodes in a given node pool, based on the demands of your workloads. There is no need to manually add or remove nodes or over-provision the node pools. The cluster autoscaler supports up to 5000 nodes running 30 Pods each.
Example:

```
gcloud container clusters create example-cluster \
  --zone us-central1-a \
  --node-locations us-central1-a,us-central1-b,us-central1-f \
  --num-nodes 2 --enable-autoscaling --min-nodes 1 --max-nodes 4
```

In Cloud Run, each revision is automatically scaled to the number of container instances needed to handle all incoming requests. When a revision does not receive any traffic, by default it is scaled into zero container instances. However, if desired, this can be changed to specify an instance to be kept idle or "warm" using the minimum instances setting. By default, container instances can scale out to 1000 instances.

5.7 Monitoring / Notifications

- SLO monitoring
  Can be used to automatically infer or custom define service-level objectives (SLOs) for applications when SLO violations occur.

- Custom metrics
  There are too many to count or display, but there are a few interesting ones: 28 day active users count, the count of 5XX (HTTP) back-end instance error codes, average latency for all disk write operations.

- Google Cloud integration
  All Google Cloud resources and services, with no additional configuration, are automatically integrated right into the Google Cloud console.

- Monitoring agent
  Allows the deployment of an open-source agent on your Google Cloud hosts and other environments to collect metrics and monitor applications.

- Logging integration
  Log data can be integrated to visualize and alert on metrics.

- Group/cluster support
  Relationships can be defined based on resource names, tags, security groups, projects, regions, accounts, and other criteria. Those relationships can be used to create targeted dashboards and topology-aware alerting policies.

- Alerting
  Alerting policies are used to notify listeners when events occur, or a particular system or custom metrics violate rules that are defined. Notifications can be received via email, SMS, Slack, PagerDuty, and more.

- Uptime monitoring
  Allows monitoring the availability of your internet-accessible URLs, VMs, APIs, and load balancers from probes around the globe. Cloud Monitoring alerting is used here for outage notifications.

5.8 Pricing

Infrastructure fee VM instance: 1 shared vCPU + 1.7 GB memory (g1-small) EUR 15.30/mo Standard Persistent Disk: 10GB EUR 0.39/mo Sustained use discount − EUR 4.59/mo Estimated monthly total EUR 11.10/mo

5.9 Creating a CI / CD pipeline

It is important to note that Jenkins with Kubernetes and Helm is used to create the CI / CD pipeline. By using a Google Cloud shell, clusters and resources are created. We are using a starter code to jump-start our CI / CD pipeline. The values should be changed according to project specifications that are going to be deployed.

Simplified version of a pipeline

A simplified version of a CI / CD pipeline can be created using only Cloud Build, the Container Registry and Cloud Run. Figure 20 shows how this would look like.

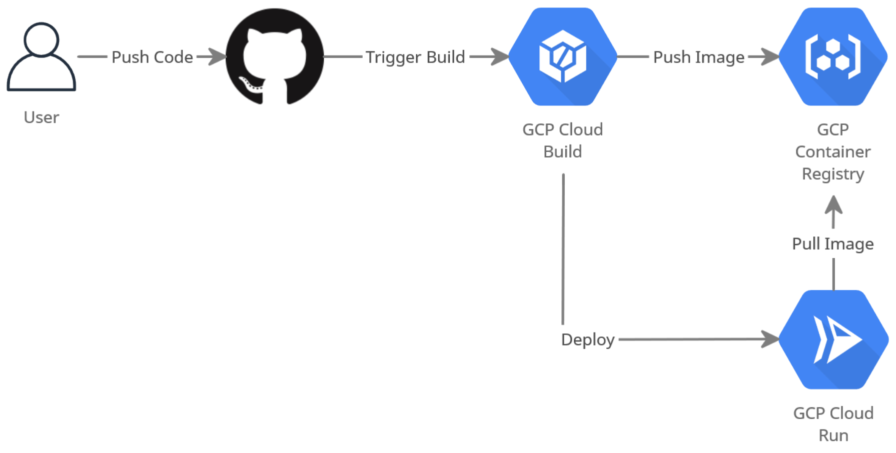


Figure 20: Jenkins pipeline

The pipeline works in the following way:

1. Developer checks in the source code to a Version Control system such as GitHub
2. GitHub triggers a post-commit hook to Cloud Build.
3. Cloud Build builds the container image and pushes to Container Registry.
4. Cloud Build then notifies Cloud Run to redeploy
5. Cloud Run pulls the latest image from the Container Registry and runs it.

Cloud Run has a few different deployment strategies: rollbacks, gradual rollouts, and traffic migration. Besides, Cloud Run also allows auto-scaling of these instances to a maximum of 1,000 with the standard quota. A drawback with this solution is that Cloud Run runs stateless containers (doesn't store data between startups) in a serverless environment.
Example of such a pipeline:

```
steps:
# Build the container image
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/PROJECT_ID/IMAGE', '.']
# Push the container image to Container Registry
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'gcr.io/PROJECT_ID/IMAGE']
# Deploy container image to Cloud Run
- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: gcloud
  args: ['run', 'deploy', 'SERVICE-NAME', '--image', 'gcr.io/PROJECT_ID/IMAGE',
    '--region', 'REGION', '--platform', 'managed']
images:
- gcr.io/PROJECT_ID/IMAGE
```

Complex pipeline

This pipeline is created using Jenkins and Kubernetes with a GKE cluster. Figure 21 shows the concept of the pipeline.

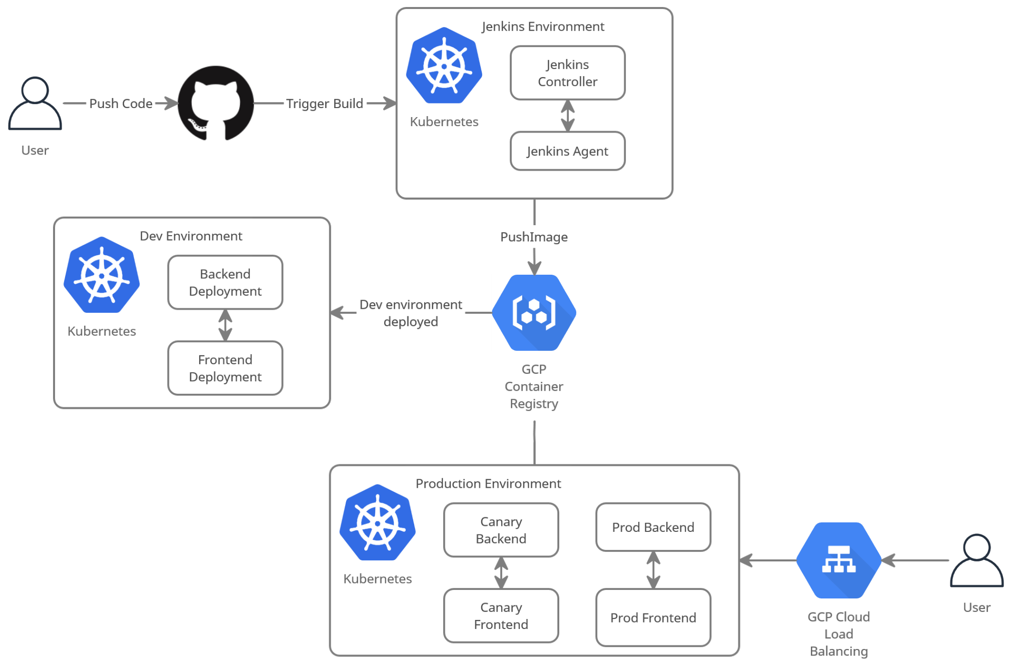


Figure 21: Pipeline

Prerequisites:

Before this pipeline can be created, the needed APIs have to be enabled: Enable the Compute Engine, Container Engine, and Container Builder APIs

CloudBuild setup:

1. Go to Cloud Build
2. Click Connect Repository
3. Select your VCS Repository (Github)
4. Specify the Name, Event (tag, normal push or pr) and Build configuration
    - The cloudbuild.yaml is primarily used to specify actions, but a custom one can be set

5. Ensure Cloud Build has access to deploy to Cloud Run if the app is going to be deployed there

Cloud shell first steps:

1. Start the Google Cloud Shell and clone the lab code repository to it.

2. When the shell is open, use the gcloud command line interface tool to set your default compute zone (other zones can be used as well):
   `gcloud config set compute/zone us-east1-d`

3. Clone the lab repository in your cloud shell, then cd into that dir:
   ```
   git clone
   https://github.com/GoogleCloudPlatform/continuous-deployment-on-kubernetes.gi
   ```

4. Go into the directory:
   `cd continuous-deployment-on-kubernetes`

5. Add required permissions, to the service account, using predefined roles:
   ```
   kubectl create clusterrolebinding jenkins-deploy
   --clusterrole=cluster-admin --serviceaccount=default:cd-jenkins
   ```

The Jenkins service account needs cluster-admin permissions so that it can create Kubernetes namespaces and any other resources that the app requires. For production use, select the individual permissions necessary and apply them to the service account individually.

Create a Kubernetes cluster:

1. Provision a Kubernetes cluster using GKE. This step can take up to several minutes to complete, if more nodes for jenkins are needed, adjust accordingly:
   ```
   gcloud container clusters create jenkins-cd
   --machine-type n1-standard-2 --num-nodes 2
   --scopes "https://www.googleapis.com/auth/source.read_write,cloud-platform"
   --cluster-version 1.15
   ```

2. Confirm that your cluster is running:
   `gcloud container clusters list`

3. Next, confirm that a connection to the cluster is possible:
   ```
   kubectl cluster-info
   ```

Install Helm:

1. Download and install the Helm binary:
   ```
   wget https://get.helm.sh/helm-v3.2.1-linux-amd64.tar.gz
   ```

2. Unzip the file to into the local system:
   ```
   tar zxfv helm-v3.2.1-linux-amd64.tar.gz cp linux-amd64/helm .
   ```

3. Add yourself as a cluster administrator in the cluster's RBAC so that you can give Jenkins permissions in the cluster:
   ```
   kubectl create clusterrolebinding cluster-admin-binding
   --clusterrole=cluster-admin
   --user=$(gcloud config get-value account)
   ```

4. Add the official stable repository:
   ```
   ./helm repo add jenkinsci https://charts.jenkins.io
   ./helm repo update
   ```

5. Ensure Helm is properly installed by running the following command:
   ```
   ./helm version
   ```

Helm is used to deploy Jenkins from the repository.

Install Jenkins:

1. Use the Helm CLI to deploy the chart with your configuration set (this can take a while):
   ```
   ./helm install cd-jenkins -f jenkins/values.yaml jenkinsci/jenkins
   --version 2.6.4 --wait
   ```

2. After that command completes, ensure the Jenkins pod goes to the Running state and the container is in the READY state:
   ```
   kubectl get pods
   ```

3. Set up port forwarding to the Jenkins UI from Cloud Shell:
   ```
   export POD_NAME=$(kubectl get pods --namespace default -l
   "app.kubernetes.io/component=jenkins-master" -l
   ```

```
"app.kubernetes.io/instance=cd-jenkins" -o
jsonpath=".items[0].metadata.name")
kubectl port-forward $POD_NAME 8080:8080 >> /dev/null &
```

4. Check that the Jenkins Service was created properly:
```
kubectl get svc
```

5. Apply the cluster-admin role to the Jenkins service account:
```
kubectl create clusterrolebinding jenkins-deploy
--clusterrole=cluster-admin --serviceaccount=default:cd-jenkins
```

Deploying the app to Kubernetes:

1. Create the Kubernetes namespaces to isolate the production deployment:
```
kubectl create ns production
```

2. Create the canary and production deployments and services (other names can be used as well, canary here is used for deployment to a subset of users):
```
kubectl --namespace=production apply -f k8s/production
kubectl --namespace=production apply -f k8s/canary
kubectl --namespace=production apply -f k8s/services
```

3. Scale up the production environment frontends (as many replicas as needed):
```
kubectl --namespace=production scale deployment gceme-frontend-production
--replicas=4
```

4. Retrieve the external IP for the production services. It can take several minutes before you see the load balancer IP address:
```
kubectl --namespace=production get service gceme-frontend
```

Add a service account to Jenkins credentials with the UI:

1. In the cloud shell, click web preview, which should open up Jenkins

2. In the Jenkins user interface, Click Credentials (or Configuration depending on the version) in the left navigation

3. Click the Jenkins link in the Credentials table
4. Click Global Credentials
5. Click Add Credentials in the left navigation
6. Select Google Service Account from metadata from the Kind dropdown list
7. Click OK

This global credential added is important if the repository is stored on Google. Otherwise, you have to add other credentials via the dropdown.

Create a Jenkins job:

1. In your cloud shell, update the Jenkinsfile script with the correct "PROJECT" environment value. It is located under the sample-app directory. Be sure to replace "REPLACE_WITH_YOUR_PROJECT_ID" with the project id. Additionally, you should also change the images that are wanted for the deployment which can be found in the "k8s" directory
2. Click the Jenkins link in the top left of the interface.
3. Click the New Item link in the left navigation.
4. Name the project sample-app, choose the Multibranch Pipeline option, and then click OK.
5. On the next page, click Add Source and select git.
6. Paste the HTTPS clone URL of your sample-app repository in Cloud Source Repositories into the Project Repository field. Replace [] with the respective values:
`https://source.developers.google.com/p/[PROJECT_ID]/r/[REPOSITORY_NAME]`
7. From the Credentials dropdown list, select the name of the credentials you created when adding your service account.
8. In the Scan Multibranch Pipeline section, select the Periodically if not otherwise run box. Set the Interval value to '1 minute'.

9. Click Save.

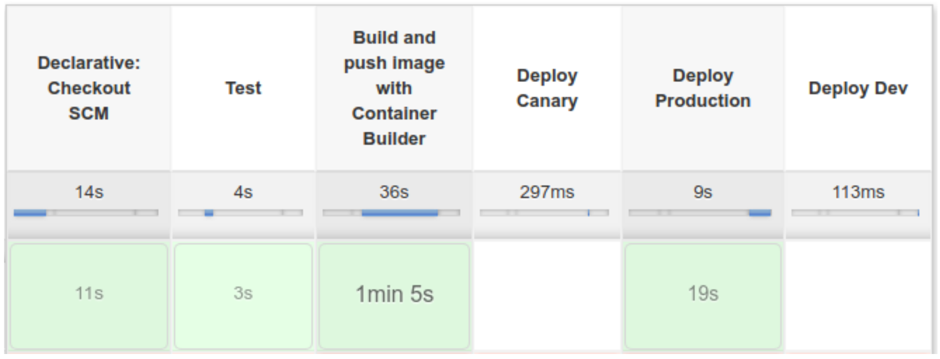
After these steps are completed, a job named "Branch indexing" runs. This meta-job identifies the branches in your repository and ensures changes haven't occurred in existing branches.

With the current version the kubernetes cloud is not added automatically, so it should be added by hand:

1. Open Jenkins
2. Click "Manage Jenkins"
3. Click "Manage Nodes and Clouds"
4. Click "Configure Clouds"
5. Add a new cloud (kubernetes)
6. Click "Save"
7. Click "Kubernetes cloud details"
8. Configure both Jenkins URL and Jenkins tunnel, example:
   `jenkins url = https://cd-jenkins:8080/`
   `jenkins tunnel = cd-jenkins-agent:50000`

Finished pipeline

The pipeline should now display the following after a successful run:



Declarative: Checkout SCM	Test	Build and push image with Container Builder	Deploy Canary	Deploy Production	Deploy Dev
14s	4s	36s	297ms	9s	113ms
11s	3s	1min 5s		19s	

Figure 22: Jenkins pipeline

6 Conclusion

6.1 Comparison and Evaluation

Scaling

AWS has extended options for Elastic Beanstalk scaling; time-based, load-based (12 different metrics!) with thresholds and different duration options. 10000 instances are the maximum amount.

Azure

Alibaba

Google Cloud Run allows for just a few different scaling options, based on concurrently (how many request each container can handle) and the amount of CPU needed to process a request. A maximum number of 1000 containers can be run at the same time as cloud run.

Kubernetes is not directly part of each provider's native CI / CD solutions, except if a pipeline is created using it. Kubernetes scaling:
"No more than 100 pods per node. No more than 5000 nodes. No more than 150000 total pods. No more than 300000 total containers."[1]

The personal complexity rating is based on how hard and long it took to create a CI / CD pipeline.

	AWS	Alibaba	Azure	Google
Complexity rating	middle	high	middle	high

Table 8: Personal complexity rating

Out of all provider, only Alibaba does not feature a dedicated build service. Thus Alibaba does not have integration with any VCS systems out of the box.

[1]https://kubernetes.io/docs/setup/best-practices/cluster-large/

	AWS	Alibaba	Azure	Google
Github	yes	no*	yes	yes
Bitbucket	yes	no*	yes	yes
Gitlab	no	no*	yes	no
Subversion	no	no*	yes	no

Table 9: VCS comparison between providers

*Alibaba does not have a build system and is therefore relying on Jenkins and other services. Jenkins however, does support all of the marked VCS options.

Regions are important for multi region CI / CD pipelines. Alibaba and Azure are constantly creating new datacenters, Alibaba has a big focus on china and is not present in a whole lot of different countries.

	AWS	Alibaba	Azure*	Google
Regions	24	22	60+	24
Zones	77	67	*	73
Countries and Territories	245	70+	140	200+

Table 10: Regions and zones comparison

*Important to note here: Azure Regions and Zones are different from AWS and others, each region can have a different amount of availability zones. There is no amount specified by Azure documentation.

The marketplace from each provider can be used to jump start certain aspects of a CI / CD pipeline. Some even feature a completed setup of Jenkins for example.

	AWS	Alibaba	Azure	Google
Marketplace / Store	yes			yes

Table 11: Store availability comparison

	AWS	Alibaba	Azure	Google
Dedicated build service	yes	no	yes	yes

Table 12: Dedicated service for building code

Docker compose was used to build 2 images; a rest API and a PostgreSQL database. AWS build time is longer due to how long the provisioning of a build VM takes. Google somehow manages almost instantly to create the needed instances for building.

	AWS	Alibaba	Azure	Google
Build time AVG	78.6s	60s - 120s*	80s	66s

Table 13: Code build time comparison

*Depends on which settings are used for the Jenkins builder, and ECS instance is used for building. Alibaba has different plans that limit bandwidth and CPU, which heavily affects build time.

The automatic creation of docker image repositories is something we did not expect from google. Creating a repository for each container is quite a tedious task, even though it is only a few clicks and entries for AWS, it still takes some time to create one.

	AWS	Alibaba	Azure	Google
Auto creation container repository	no	no	yes	yes

Table 14: Automatic docker container repository creation comparison

Since not every provider has comparable services, we just list the different up-time guarantees for each service used to create the pipeline. AWS single EC2 instances (Table 15) have the lowest up-time guarantee, the reason was not stated by AWS.

AWS service	Uptime
Code Build	99.99%
Code Build	99.99%
EC2, EBS, ECS, Fargate	99.99%
Single EC2	90%
RDS multi AZ	99.5%

Table 15: Uptime guarantees AWS

Azure service	Uptime
Azure DevOps	99.9%
Container Registry	99.5%
App Service	99.95%
Azure Database for PostgreSQL	99.5%

Table 16: Uptime guarantees Azure

Google service	Uptime
Cloud Build	99.95%
Zonal Cluster	99.5%
Regional Cluster	99.95%
Cloud SQL	99.5%

Table 17: Uptime guarantees Google

Cost comparisons are not 100% accurate and are a rough estimate. While we used different provider services, it is good to estimate how much a CI / CD pipeline with the lowest settings can cost. In table 18, google is significantly higher as the Kubernetes nodes in a cluster cost quite a bit. The smallest PostgreSQL database already costs roughly 35$ per month, whereas the AWS smallest database costs roughly 14$. For the calculations, storage and artifact registry was not included, though, for AWS and Google, the estimate per GB is roughly 0.10$.

6.2 Key Takeaways

AWS and Azure offer the easiest solutions for a quick CI / CD pipeline creation. While it can take a while to figure out the different needed per-

	AWS	Alibaba	Azure	Google K8S	Google Cloud Run
Overall / Month	42$	95$	104$	93$	71$

Table 18: Cost comparison overall for the CI CD pipeline

missions for each part, setup of additional resources becomes quite easy and fast. In a matter of minutes, pipelines can be created from scratch. Alibaba and Google have severe limitations for individuals that are not familiar with Jenkins/Spinnaker and Kubernetes as all or some are required to create a pipeline. The commands and tutorials are not up to date, and a lot of different tweaks had to be made as none of the available guides from the official sources worked out of the box. This creates another barrier and makes it quite hard to create pipelines on the go. If a fast and reliable pipeline creation is wanted, AWS and Azure are recommended.

It should be noted that the student accounts for AWS and Azure have some limitations. An AWS student account can not be used to create a pipeline, and a personal account is needed.

6.3 E-Learning related Questions

1. Automated Infrastructue Provisioning/(Infrastructure-as-Code). Wie wurde im vorliegenden Projekt Automated Infrastructure Provisioning berücksichtigt?
   AWS Elastic Beanstalk uses AWS Cloud Formation in the background which is used to automatically provision needed instances. Additionally, Elastic Beanstalk environment configuration allows the creation of templates that can be reused.
   Azure allows the use of JSON templates for resources. These contain for example the kind (app, container, linux) and the properties of the specific resource.
   For Alibaba and Google Kubernetes was used which can be used to provision needed instances.

2. Skalierbarkeit. Wie wurde im vorliegenden Projekt Skalierbarkeit berücksichtigt?
   The databases and instances used for the rest API are automatically scaled using the provided services.

3. Ausfallssicherheit. Wie wurde im vorliegenden Projekt Ausfallssicherheit berücksichtigt?
   Each service guaranteed uptimes were collected, but no extra steps for the CI/CD pipeline were taken to ensure realiability.

4. NoSql. Welchen Beitrag leistet NoSql in der vorliegenden Problemstellung?
   NoSql was not part of this research.

5. Replikation. Wo nutzen Sie im gegenständlichen Projekt Daten-Replikation?
   Each cloud provider offers replication for databases out of the box. For the CI / CD pipeline itself, data replication is not needed.

6. Kosten. Welche Kosten verursacht Ihre Lösung? Welchen monetären Vorteil hat diese Lösung gegenüber einer Nicht-Cloud-Lösung?
   See Table 18.