Issues in Information Systems
Volume 22, Issue 1, pp. 136-148, 2021

https://doi.org/10.48009/1_iis_2021_136-148

Implementation of DevOps paradigm to deployment and provisioning of microservices

Abhijit Sen, *Kwantlen Polytechnic University, abhijit.sen@kpu.ca*
Ivan Skrobot, *British Columbia Institute of Technology, ivanscrobot@gmail.com*

Abstract

Microservices architecture is widely used in the industry to deploy applications in terms of well-defined services along with DevOps paradigm to frequently release features. This paper describes how to incorporate some widely used DevOps tools for the deployment of micro-services. Amazon Web Services (AWS) Serverless architecture is used for the deployment of microservices using AWS Elastic Container Service. The process of deployment and provisioning of microservices using main features of AWS Elastic Container Service will be demonstrated and discussed in this paper with a simple example. We have applied and integrated DevOps tools/technologies to manage and deploy microservices. The example project is one of many projects' students use to master DevOps skills and practices in the course "DevOps Principles and Practices" using current technologies.

Keywords: Microservices, AWS, DevOps, Container

Introduction

Software industry is increasingly applying and using principles of DevOps for rapid, frequent delivery, deployment, and release of software. DevOps is based on the tenets of Continuous planning, Collaborating development, Continuous Testing, Release and Deployment, and Continuous monitoring. DevOps is normally implemented using eight stages: Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor (Sen, 2021). To facilitate and automate implementing the various tasks associated with each of these stages, number of tools are developed (Barahalikar, 2020) and used. Organizations are also increasingly embracing Cloud Technology to house applications, data and platform. There are also shifts in the system design architecture, where monolithic architecture is being replaced by micro services architecture. In micro services architecture in contrast with monolithic architecture, a single application is set up as number of independently developed and deployed services. These services may use different programming languages, different data stores, and different platforms (Flower, Lewis, 2014, Redhat). A monolithic application on the other hand is designed as a single unit, and generally uses same programming languages. In this paper Amazon Web Services (AWS) will be integrated with DevOps tools for managing and deployment of microservices.

This paper describes one student project from course DevOps Principles and Practices given for Information Technology undergraduate students. The aim of the course is to equip students with necessary skills in DevOps principles and practices using industry standard tool chains.

Literature survey

DevOps paradigm comprises of a set of principles and practices that integrates software development (Dev) and information technology operations (Ops) to enhance application delivery frequency of high quality software product. To this end various DevOps models have been suggested, one such model showing eight stages is depicted (Bluestone; Drew) in Figure 1. Wide variety of tools available to facilitate each of these stages (Sen,2021; Barahalikar, 2020). The article (Dennis, 2020) discusses the DevOps tools commonly used in industry. It is always a challenge to select the proper tool chains appropriate for the project as more and more new tools and artifacts are being developed.

Many organizations are moving towards development of micro services architecture, where a single application is split into set of smaller, interconnected services (Richardson, 2015). Each microservice consists of simple business logic along with various adapters. These microservices may interact with other microservices or with various application's clients. Other microservices might implement a web UI. At runtime, each instance of microservice may be imaged as a Docker container which includes everything needed to run an application: code, runtime, system tools, system libraries and settings (Richardson, 2015).

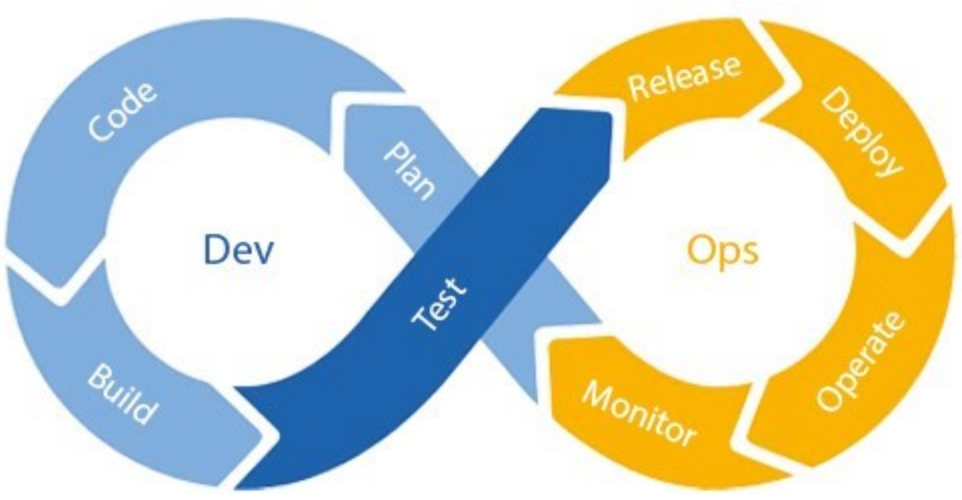


Figure 1. DevOps Life Cycle Source: https://intercept.cloud/en/news/what-is-azure-devops/

As shown in Figure 1, DevOps consists of eight stages- Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor., where DEV consists of {Plan, Code, Build, Test, Release, and Deploy} segments, and Ops consist of {Operate, and Monitor} segments.

In this paper attempt is made to integrate DevOps principles with micrservice architecture using some popular tool sets. To keep the paper concise and to emphasize implementation, planning and management parts of the project will not be described.

The rest of the paper is organized as per the following manners. The various tools available for microservices and DevOps are discussed in "DEVOPS TOOLS FOR MICRO SERVICES" section. The actual application developed is discussed in section "DESCRIPTION OF APPLICATION". Section titled "TOOLS USED FOR APPLICATION" discusses selected tools used for application development and deployment. The process of application deployment is discussed in section titled "APPLICATION DEPLOYMENT" followed by conclusion in section titled "CONCLUSIONS".

DevOps tools for micro services

There are wide variety of DevOps tools that can be used to build and deploy micro services. These tools can be classified as follows:

- API Testing and Microservices Testing tools: allow automated testing for different stages of CI/CD pipelines. The most popular tools are Postman, Tyk, and API Fortress, Goreplay, WireMock
- Orchestration Tools: such as Kubernetes, Docker Swarm, OpenShift, AWS EKS manages micro services workflows or processes.
- Monitoring Tools: monitor containers and services for performance and health, move or terminate services, generate alerts based on service performance. There are various monitoring systems and tools that specialize in microservices such as Prometheus, cAdviser, Sensu, Grafana.
- Serverless Tools: provide functionality and platforms for micro services "as-a-service", eliminating the need to deploy and maintain infrastructures. Leading cloud vendors offer serverless computation (AWS Lambda, Google Cloud Function, Azure Automation, Azure App Service), serverless Kubernetes Clusters (AWS EKS, Google Kubernetes Engine, Azure Serverless Kubernetes, Azure Red Hat OpenShift), serverless containers (AWS Elastic Container Service, Azure Container Instances). Most of the serverless tools can be conFigd as a code, and therefore are useful for DevOps practices.

Description of application

The purpose of the example application is to demonstrate a micro service application that provides methods to manage marks for students for different tasks such as assignments, seminar, project and examination marks. The example application implements several HTTP methods that allow requesting, modifying, and deleting student:

- **GET method** returns information about all tasks or a specific task. Data is represented in JSON format.
- **POST method** adds a new set of task.
- **PUT method** modifies a data for a specific task
- **DELETE method** deletes a set of tasks.

Data in the example is represented in JSON format. The details of the methods are described below:

- **GET method.** Return all data about assignments currently stored in the application memory. Persistent storage is not used in the example application. Example of request: "http://34.122.33.230/json".
- **Example of response:**

```
{
"assignments":
{
     "Assignment 1": "100%",
     "Assignment 2": "95%",
     "Assignment 3": "Not Submitted",
     "Final project": "Not Submitted",
     "Seminar": "Not Submitted"
  }
}
```

- **GET method**. Returns data about a specific assignment. Example of request: "http://34.122.33.230/json/assignments/Assignment 1".

- **Example of response:**

```
{
  "res": "100%"
}
```

- **POST method.** Add new set of assignments. Example of request: "http://34.122.33.230/json/exams", with BODY {"exam 1":"test", "exam 2":"coding"}
- **Example of response:**

```
{
  "message": "Collection created"
 }
```

- **PUT method.** Modify information about a record in a set of assignments. Example of request: "http://34.122.33.230/json/assignments/Assignment 3", with BODY { "new":"100%"}
- **Example of response:**

```
{ "res":
  {
      "Assignment 1": "100%",
      "Assignment 2": "95%",
      "Assignment 3": "100%",
      "Final project": "Not Submitted",
      "Seminar": "Not Submitted"
},
   "exams": {
      "exam 1": "test",
      "exam 2": "coding"
   }
}
```

- **DELETE method.** Delete a set of assignments. Example of request: http://34.122.33.230/json/exams .
- **Example of response:**

```
{
   "assignments": {
      "Assignment 1": "100%",
      "Assignment 2": "95%",
      "Assignment 3": "100%",
      "Final project": "Not Submitted",
      "Seminar": "Not Submitted"
   },
```

The application is developed using Python language, using Flask micro web application framework.

Tools used for application

The application is developed using the following tools:

- Amazon Elastic Container Service (Amazon ECS) – is used to run containers.

- AWS Fargate - provides container management service AWS CloudWatch - monitors AWS applications and resources in the cloud, in real time (Nick,2020).
- Flask Framework for Python - provides tools, libraries and technologies to build a web application (Hunt-Walker, 2020).
- Docker is used to build and share containerized applications
- Postman Desktop Agent tool is used to send HTTP requests to the web application (Lane, 2020).

Application deployment

This section describes the deployment process of the application using AWS ECS service. The deployment process consists of the following steps. **Note all the figures are shown in Appendix**.:

- Select cluster template: Clusters group tasks in logical groups. Clusters can be based on AWS EC2 (virtual machines) or serverless Fargate infrastructure. As shown in Fig 2, Fargate serverless infrastructure is chosen for this application from number of different templates.
- ConFig cluster: A cluster can be placed in a new virtual private cloud segment, or existing VPC's will be used later for the cluster's tasks. Tags can be applied for grouping AWS resources in logical sets. AWS CloudWatch monitoring tool can be integrated with the cluster. The configuration is shown in Fig 3.
- Define tasks. See Fig 4 for defining task template.
- Select Launch Type- As shown in Fig 5 Fargate is selected as the launch type for our application.
- Task Definition- As shown in Fig 6 Task definitions describe one or several containers (up to ten) that form an application. Task definitions specify vCPU and memory resources, open ports, private networks, data volumes, and other parameters of containers.
- Define Container – As shown in Fig 7, container name and a Docker image name must be specified
- Create new task in the cluster- Fig 8 shows the configuration of the new task in the cluster which is ready to be run
- ConFig Task to run- Fig 9 shows how the task to be conFigd for running by specifying launch type (Fargate or EC2), task definition, the number of tasks, virtual private clouds, and subnets. Fig 10 specifies the network configuration along with security groups
- Run the task – Fig 11shows After being run, tasks are presented in the "Tasks" tab of cluster definition. The detailed state of a new task is available in a separate view as shown in Fig 12. Among other information, this view provides an IP address of the created container.

At this stage, the application is ready for work. It can be called by sending a get request from a web browser (Fig 13). The application shows the current values of various evaluation items such as assignment, seminar etc. Fig 14, shows post request is sent from Postman with additional items. Using get request one can view the updated result in a web browser as shown in Fig 15.

Conclusions

Microservices architecture allows more flexible development of business processes using functionally independent service. The focus of microservice architecture is to build business functionalities. As such, microservices need to be organized, managed and designed based on specific business requirements. With microservices and DevOps organizations will be able to deliver and deploy new features and services much faster to the customers. With microservices, business processes can be easily automated and any required change in business process can be implemented by modifying the associated micoservices. These independent services can be easily managed, tested and monitored. Number of testing tools such as Postman are available to set up automated tests for microservices. However, increasing number of services may render the process of managing, building, testing and maintenance more complex. With the availability of many DevOps tools many of the complexities of managing, maintaining, testing and deployment of microservices can be mitigated. In particular, several SaaS products offer the functionality of a serverless platform for microservices for the deployment process. One of those products, AWS ECS, was described and tested in this paper. The implementation of the sample micro services using AWS ECS might significantly facilitate deployment and maintenance of microservices, thereby making AWS ECS a useful service for DevOps engineers. Docker was used to share containerized application, and Postman agent running locally on desktop is used as an agent for making API calls on user's behalf.

From test to production stages, AWS ECS can be used for different environments, AWS ECS can significantly reduce the time and efforts needed to deploy microservices without recourse to manual deployment and configuration of a server. Moreover, AWS provides high level security and reliability without any additional efforts. AWS offers ECS integration with its other services, such as Elastic LoadBalancer and Identity Access Management. It facilitates the deployment of complex multi-component applications in AWS Cloud. The students in the class DevOps Principles and Practices gained valuable industry sought after skill sets. Many students who mastered the skills have obtained related employment in software development.

References

Amazon Elastic Container Service. https://aws.amazon.com/ecs/features/, Accessed 2021/02/12.

AWS CloudWatch, https://aws.amazon.com/cloudwatch/, Accessed 2021/02/12.

Barahalikar, S. (2020). DevOps Tools and Technologies to Manage Microservices https://www.hcltech.com/blogs/devops-tools-and-technologies-manage-microservices, last accessed 2021/02/12.

Bluestone, D, Drew, G. Electrify Web Developments with DevOps, https://www.cyber-duck.co.uk/insights/resources/electrify-web-development-with-devops

Cloud computing with AWS. https://aws.amazon.com/what-is-aws/, last accessed 2021/02/12.

Dennis, D. (2020). (DevOps Tools Every Engineer Should Know. https://dev.to/daviddennis02/devops-tools-every-engineer-should-know-hf2, last accessed 2021/02/12.

Docker. https://www.docker.com/, last accessed 2021/02/12.

Flower, M., Lewis, J. (2014). Microservices. A definition of this new architectural term,: https://martinfowler.com/articles/microservices.html#ComponentizationViaServices, last accessed 2021/02/12.

Hunt-Walker, N. (2020). An introduction to the Flask Python web app framework, https://opensource.com/article/18/4/flask, last accessed 2021/02/12.

Lane, K. (2020). Introducing the Postman Agent: Send API Requests from Your Browser without Limits, https://blog.postman.com/introducing-the-postman-agent-send-api-requests-from-your-browser-without-limits/, last accessed 2021/02/12.

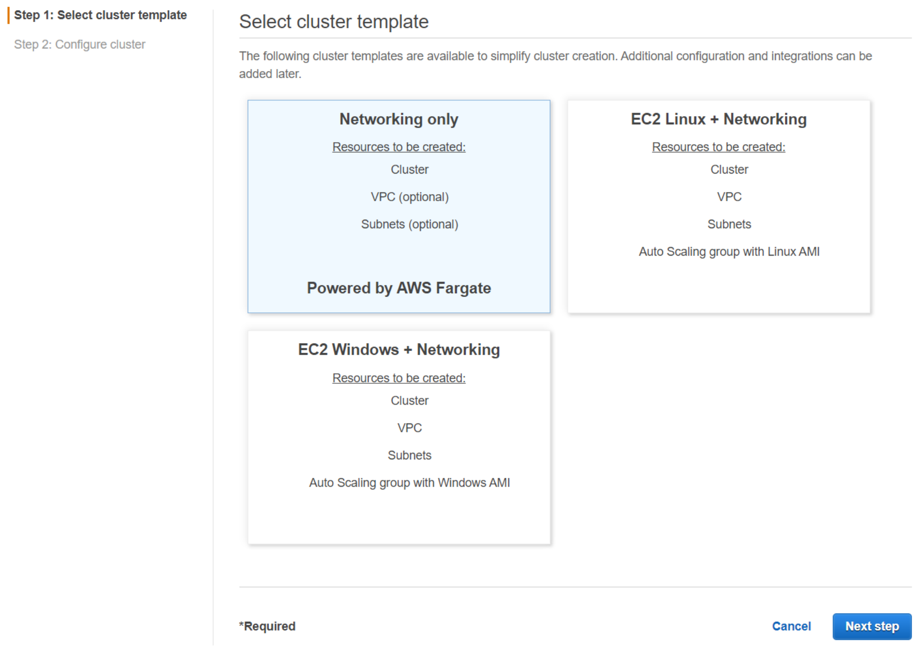
Nick. (2020). How to Get Started with AWS Fargate: Tutorial and Next Steps, https://blog.iron.io/get-started-with-aws-fargate-tutorial/, last accessed 2021/02/12.

RedHat: What are microservices?: https://www.redhat.com/en/topics/microservices/what-are-microservices, Accessed 2021/02/12.

Richardson, C. (2015). Introduction to Microservices, https://www.nginx.com/blog/introduction-to-microservices/, Accessed 2021/02/12.

Sen, A. (2021). DevOps, DevSecOps, AIOPS- Paradigms to IT Operations. Evolving Technologies for Computing, Communication and Smart World, Pages 211-221, Springer

APPENDIX



Select cluster template

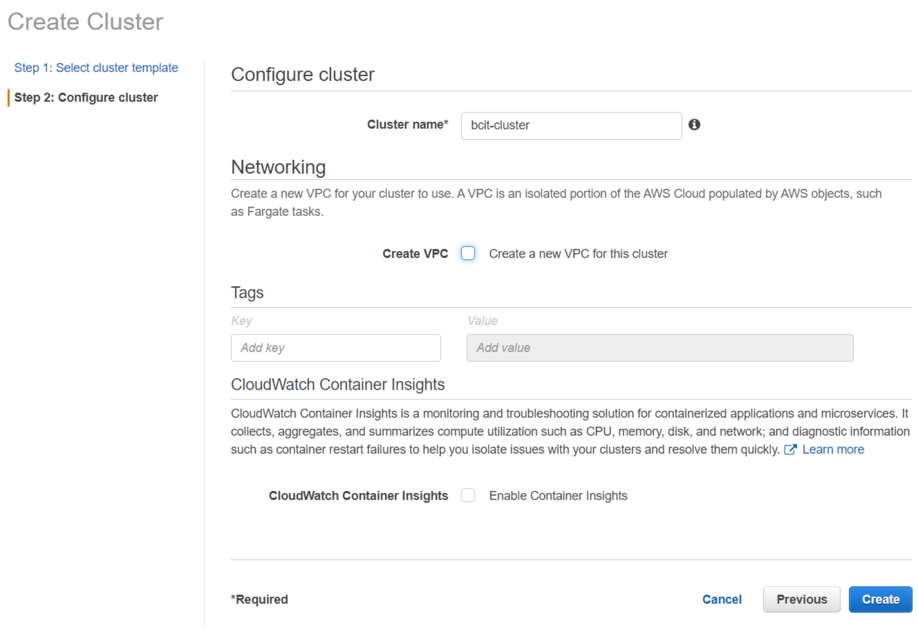
The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

Networking only
Resources to be created:
Cluster
VPC (optional)
Subnets (optional)
Powered by AWS Fargate

EC2 Linux + Networking
Resources to be created:
Cluster
VPC
Subnets
Auto Scaling group with Linux AMI

EC2 Windows + Networking
Resources to be created:
Cluster
VPC
Subnets
Auto Scaling group with Windows AMI

Figure 2. Select cluster template



Create Cluster

Step 1: Select cluster template
Step 2: Configure cluster

Configure cluster

Cluster name* bcit-cluster

Networking

Create a new VPC for your cluster to use. A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Fargate tasks.

Create VPC ☐ Create a new VPC for this cluster

Tags

Key: Add key — Value: Add value

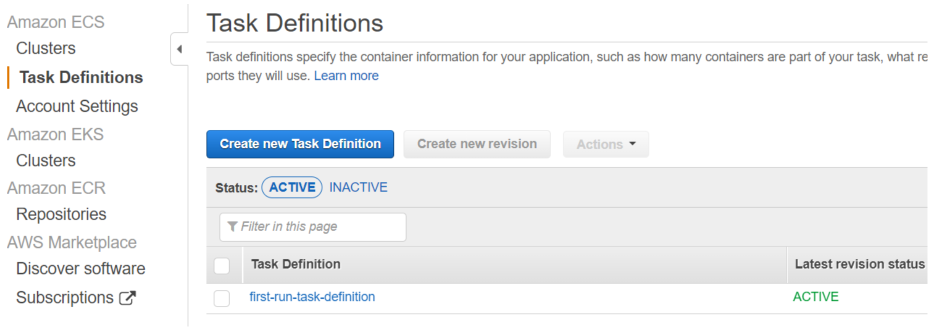
CloudWatch Container Insights

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices. It collects, aggregates, and summarizes compute utilization such as CPU, memory, disk, and network; and diagnostic information such as container restart failures to help you isolate issues with your clusters and resolve them quickly. Learn more

CloudWatch Container Insights ☐ Enable Container Insights

*Required — Cancel — Previous — Create

Figure 3. ConFig cluster



Task Definitions

Task definitions specify the container information for your application, such as how many containers are part of your task, what re ports they will use. Learn more

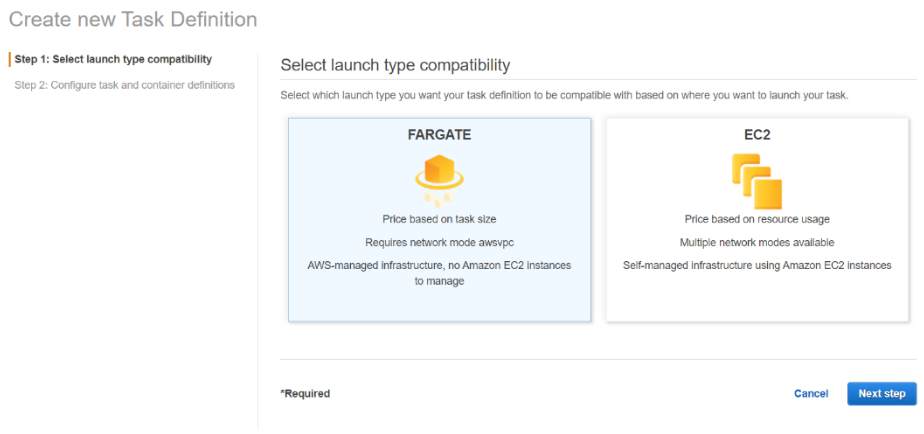
Create new Task Definition — Create new revision — Actions

Status: ACTIVE INACTIVE

Filter in this page

Task Definition	Latest revision status
first-run-task-definition	ACTIVE

Figure 4. Task Definitions



Create new Task Definition

Step 1: Select launch type compatibility
Step 2: Configure task and container definitions

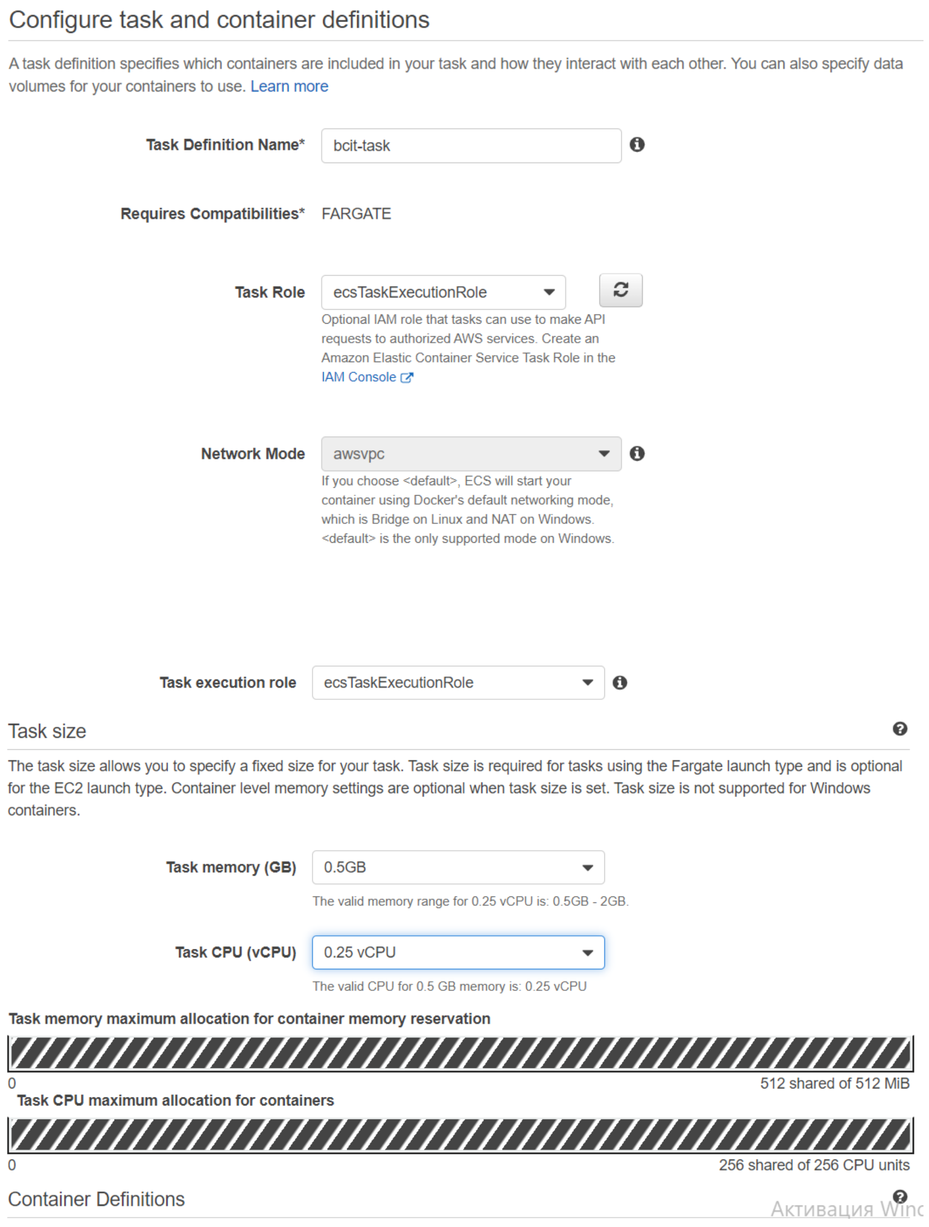
Select launch type compatibility

Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

FARGATE	EC2
Price based on task size	Price based on resource usage
Requires network mode awsvpc	Multiple network modes available
AWS-managed infrastructure, no Amazon EC2 instances to manage	Self-managed infrastructure using Amazon EC2 instances

*Required — Cancel — Next step

Figure 5. Select Launch Type



Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other. You can also specify data volumes for your containers to use. Learn more

Task Definition Name* bcit-task

Requires Compatibilities* FARGATE

Task Role ecsTaskExecutionRole
Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the IAM Console

Network Mode awsvpc
If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. <default> is the only supported mode on Windows.

Task execution role ecsTaskExecutionRole

Task size

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (GB) 0.5GB
The valid memory range for 0.25 vCPU is: 0.5GB - 2GB.

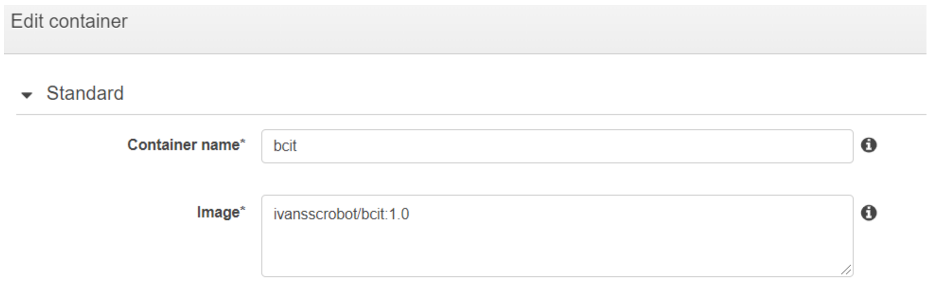
Task CPU (vCPU) 0.25 vCPU
The valid CPU for 0.5 GB memory is: 0.25 vCPU

Task memory maximum allocation for container memory reservation
0 512 shared of 512 MiB

Task CPU maximum allocation for containers
0 256 shared of 256 CPU units

Container Definitions

Figure 6. Task Definition



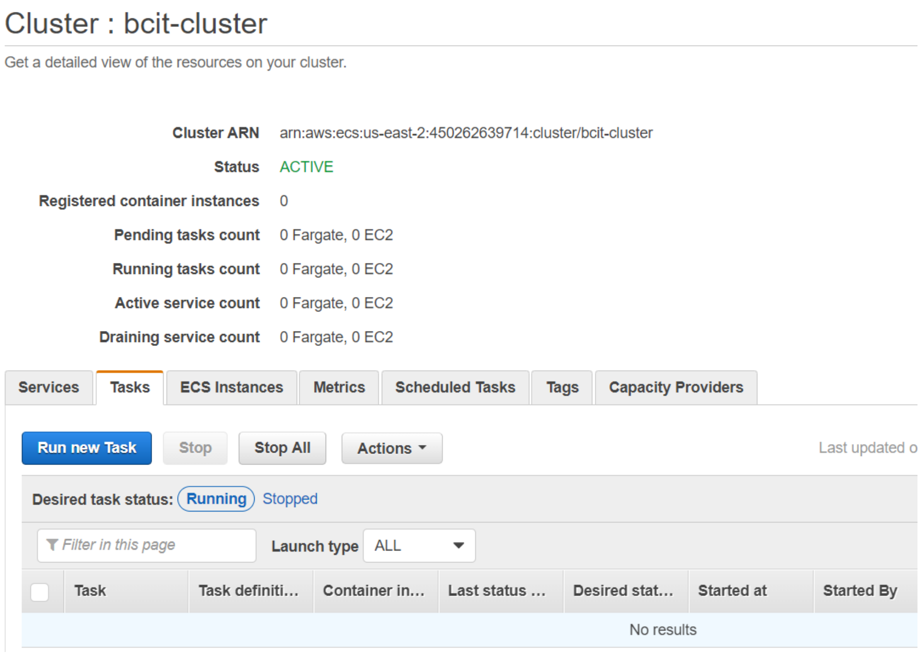
Edit container

Standard

Container name* bcit

Image* ivansscrobot/bcit:1.0

Figure 7. Container Definition



Cluster : bcit-cluster

Get a detailed view of the resources on your cluster.

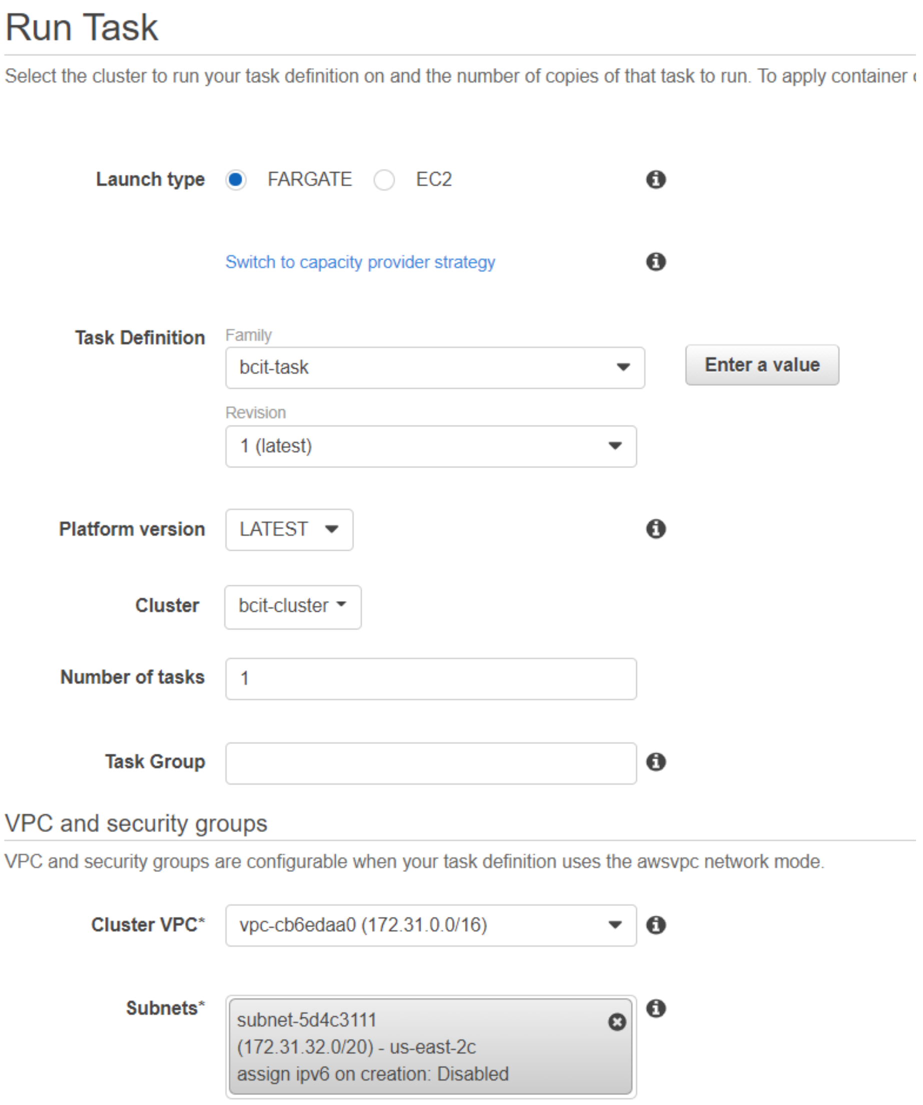
Cluster ARN	arn:aws:ecs:us-east-2:450262639714:cluster/bcit-cluster
Status	ACTIVE
Registered container instances	0
Pending tasks count	0 Fargate, 0 EC2
Running tasks count	0 Fargate, 0 EC2
Active service count	0 Fargate, 0 EC2
Draining service count	0 Fargate, 0 EC2

Desired task status: Running Stopped

Launch type ALL

Task	Task definiti...	Container in...	Last status ...	Desired stat...	Started at	Started By
No results						

Figure 8. Create new task in the cluster


Run Task

Select the cluster to run your task definition on and the number of copies of that task to run. To apply container

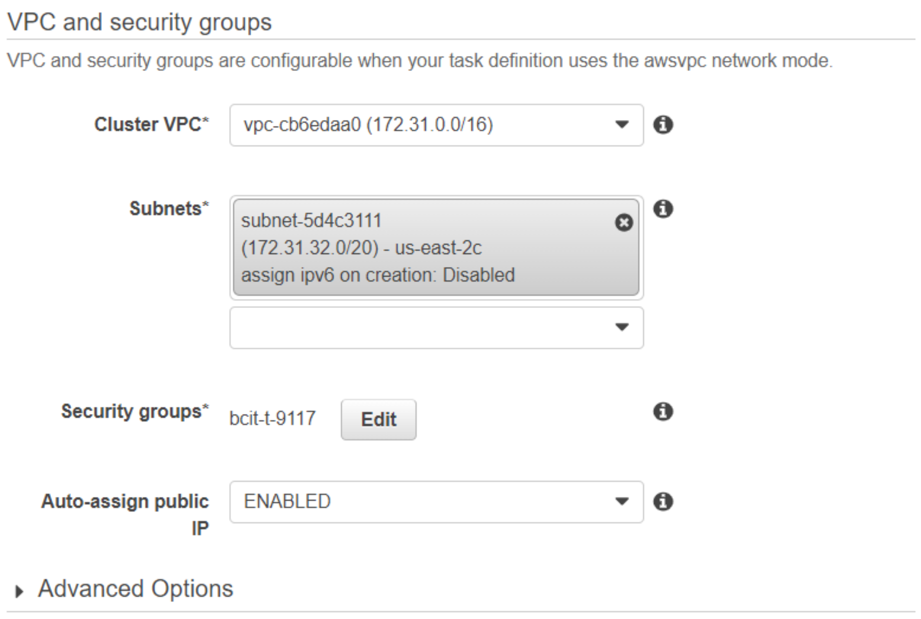
Launch type	[✓ FARGATE] EC2
	Switch to capacity provider strategy
Task Definition	Family: bcit-task; Revision: 1 (latest)
Platform version	LATEST
Cluster	bcit-cluster
Number of tasks	1
Task Group	

VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC*	vpc-cb6edaa0 (172.31.0.0/16)
Subnets*	subnet-5d4c3111 (172.31.32.0/20) - us-east-2c assign ipv6 on creation: Disabled

Figure 9. ConFig Task to run

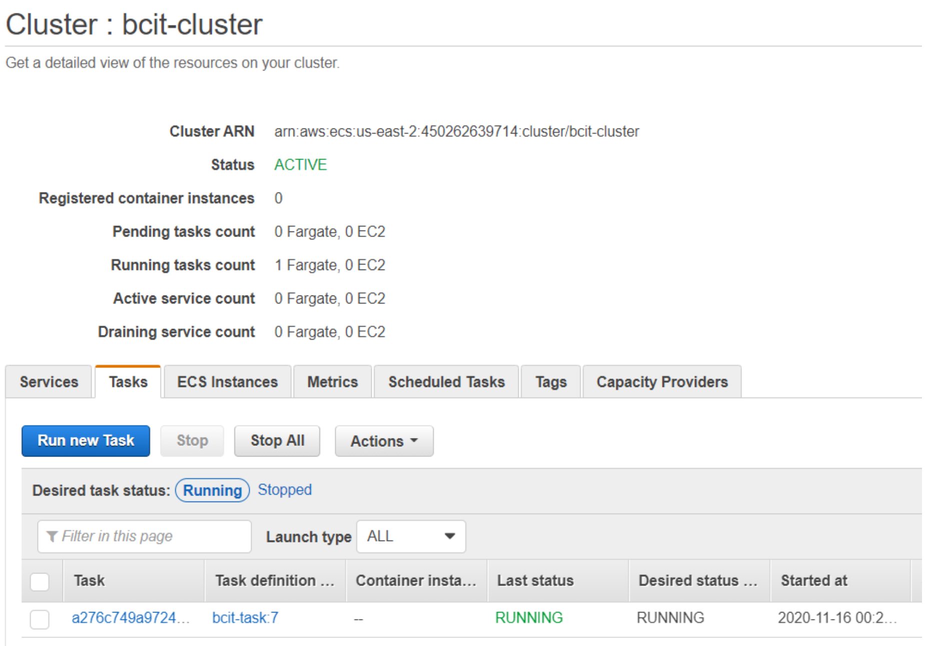


VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC*	vpc-cb6edaa0 (172.31.0.0/16)
Subnets*	subnet-5d4c3111 (172.31.32.0/20) - us-east-2c assign ipv6 on creation: Disabled
Security groups*	bcit-t-9117
Auto-assign public IP	ENABLED

Figure 10. Network Configuration for the new task



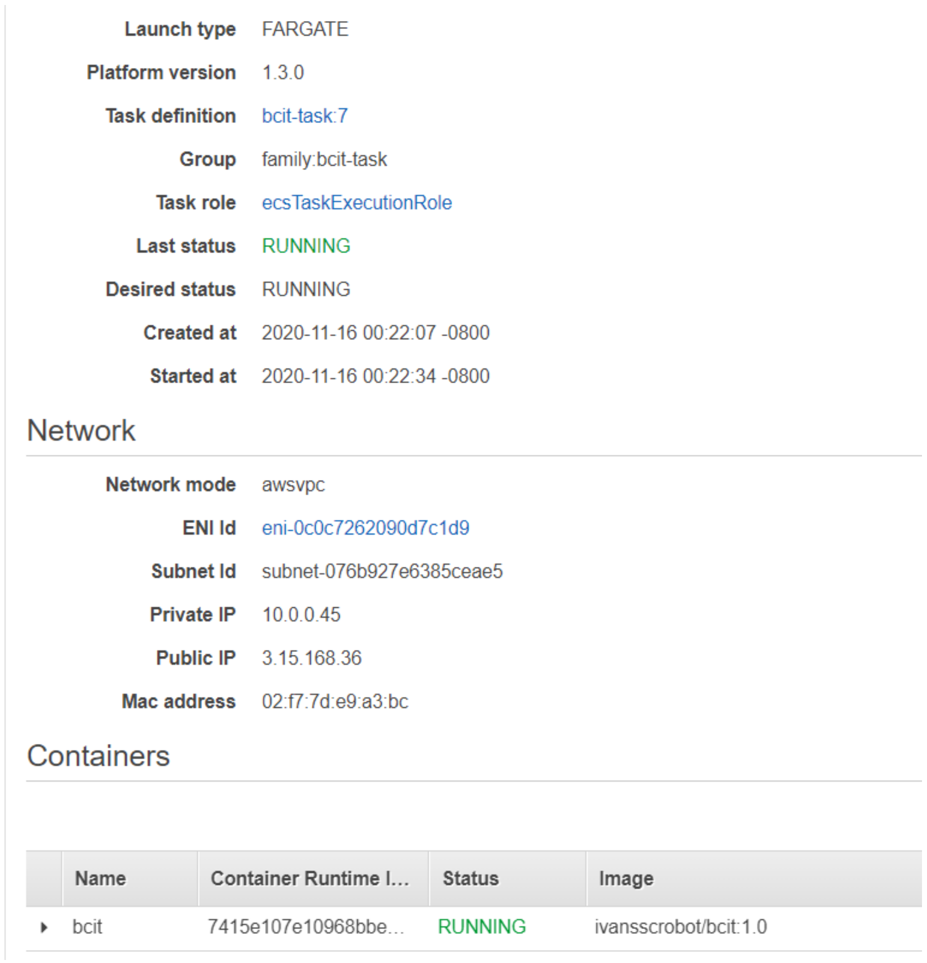
Cluster : bcit-cluster

Get a detailed view of the resources on your cluster.

Cluster ARN	arn:aws:ecs:us-east-2:450262639714:cluster/bcit-cluster
Status	ACTIVE
Registered container instances	0
Pending tasks count	0 Fargate, 0 EC2
Running tasks count	1 Fargate, 0 EC2
Active service count	0 Fargate, 0 EC2
Draining service count	0 Fargate, 0 EC2

Task	Task definition ...	Container insta...	Last status	Desired status ...	Started at
a276c749a9724...	bcit-task:7	--	RUNNING	RUNNING	2020-11-16 00:2...

Figure 11. Running of the new task



Launch type	FARGATE
Platform version	1.3.0
Task definition	bcit-task:7
Group	family:bcit-task
Task role	ecsTaskExecutionRole
Last status	RUNNING
Desired status	RUNNING
Created at	2020-11-16 00:22:07 -0800
Started at	2020-11-16 00:22:34 -0800

Network

Network mode	awsvpc
ENI Id	eni-0c0c7262090d7c1d9
Subnet Id	subnet-076b927e6385ceae5
Private IP	10.0.0.45
Public IP	3.15.168.36
Mac address	02:f7:7d:e9:a3:bc

Containers

	Name	Container Runtime I...	Status	Image
▸	bcit	7415e107e10968bbe...	RUNNING	ivansscrobot/bcit:1.0

Figure 12. State of the new task