

Lecture #11

Serverless

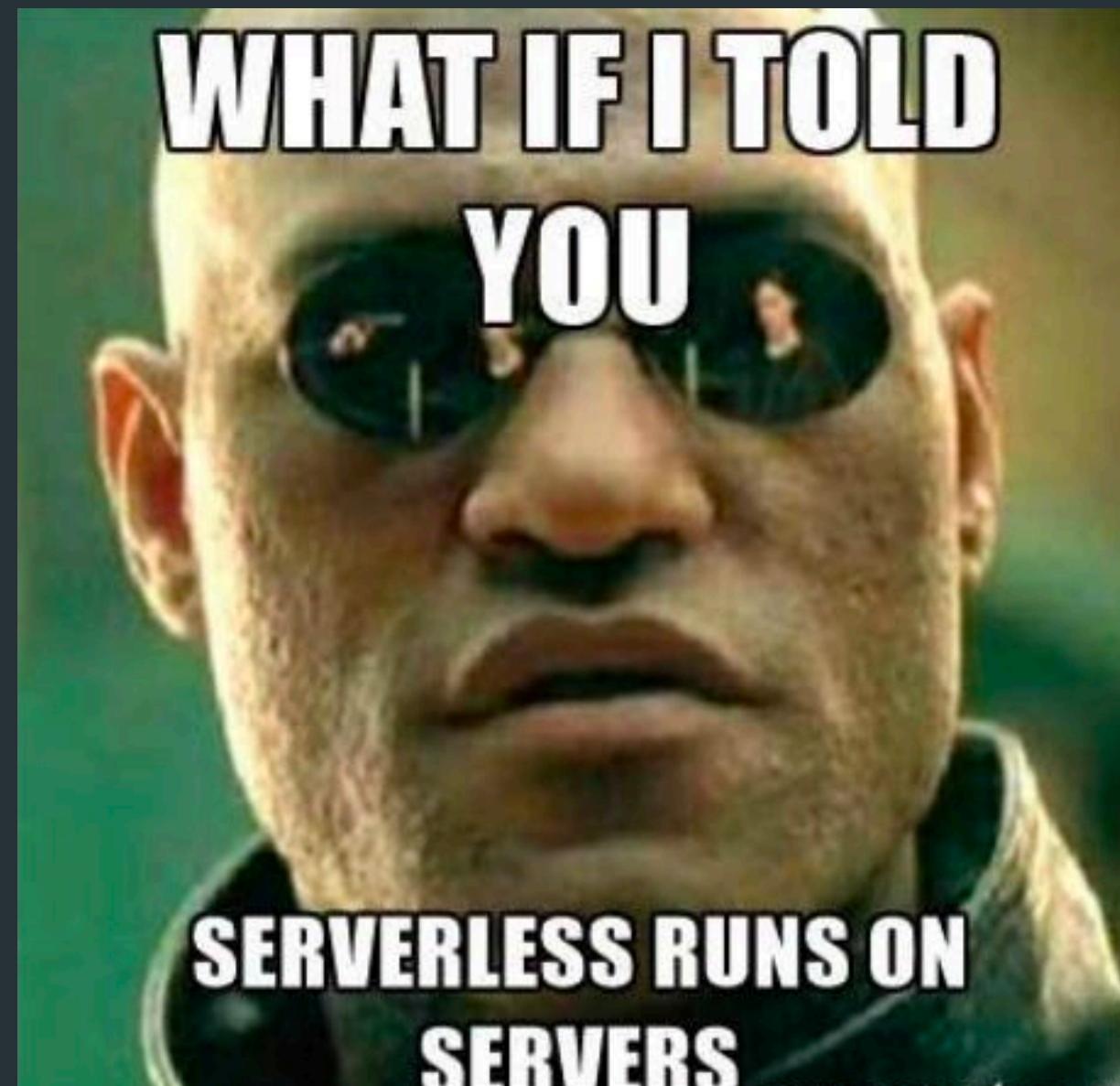
WSMT2023

Introduction to Serverless

Serverless computing is a cloud computing model in which the cloud provider dynamically manages the allocation of machine resources. This eliminates the need for:

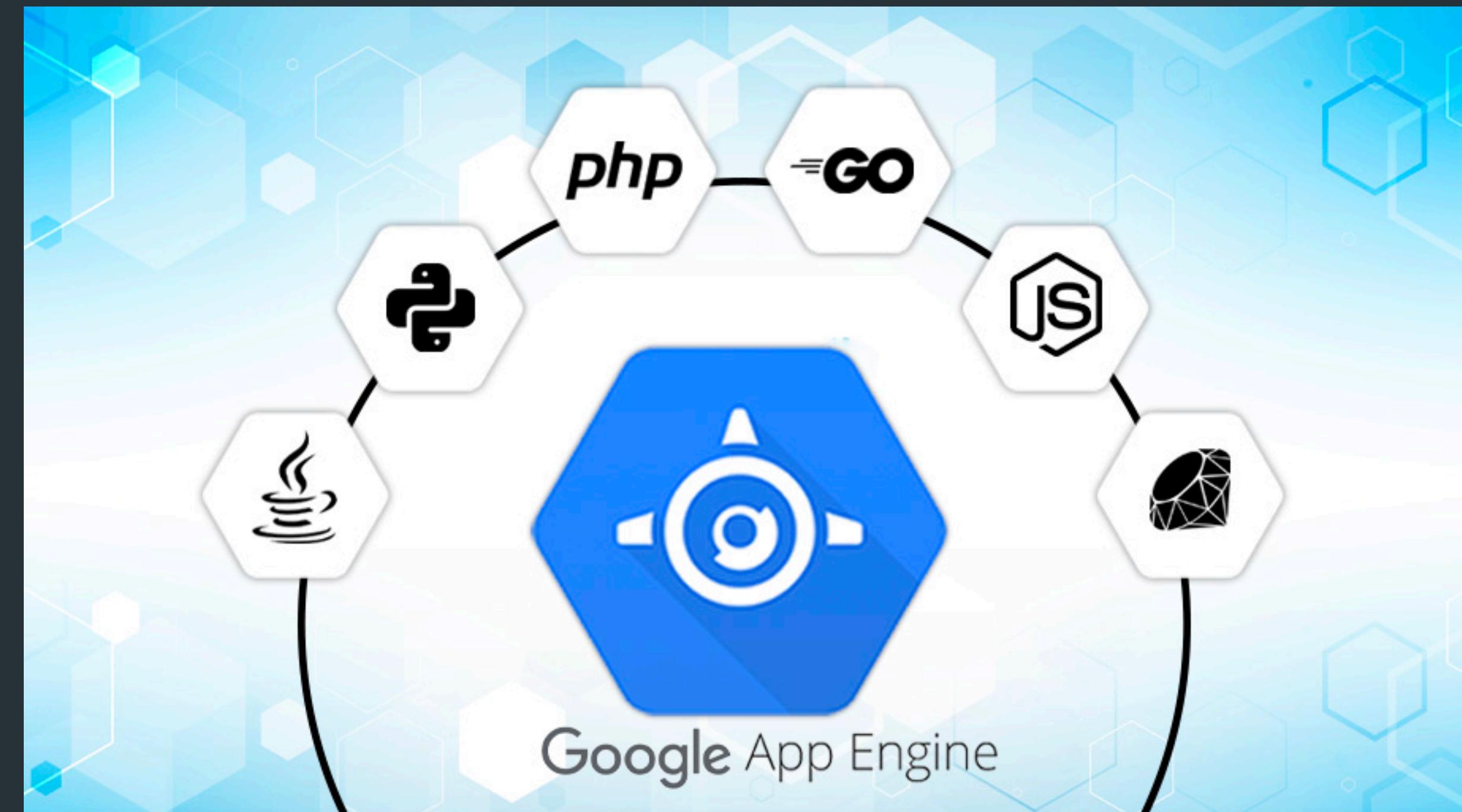
- Server administration
- Provisioning
- Maintenance

Freeing up developers to focus on their applications.



History

- 2008: Google App Engine.



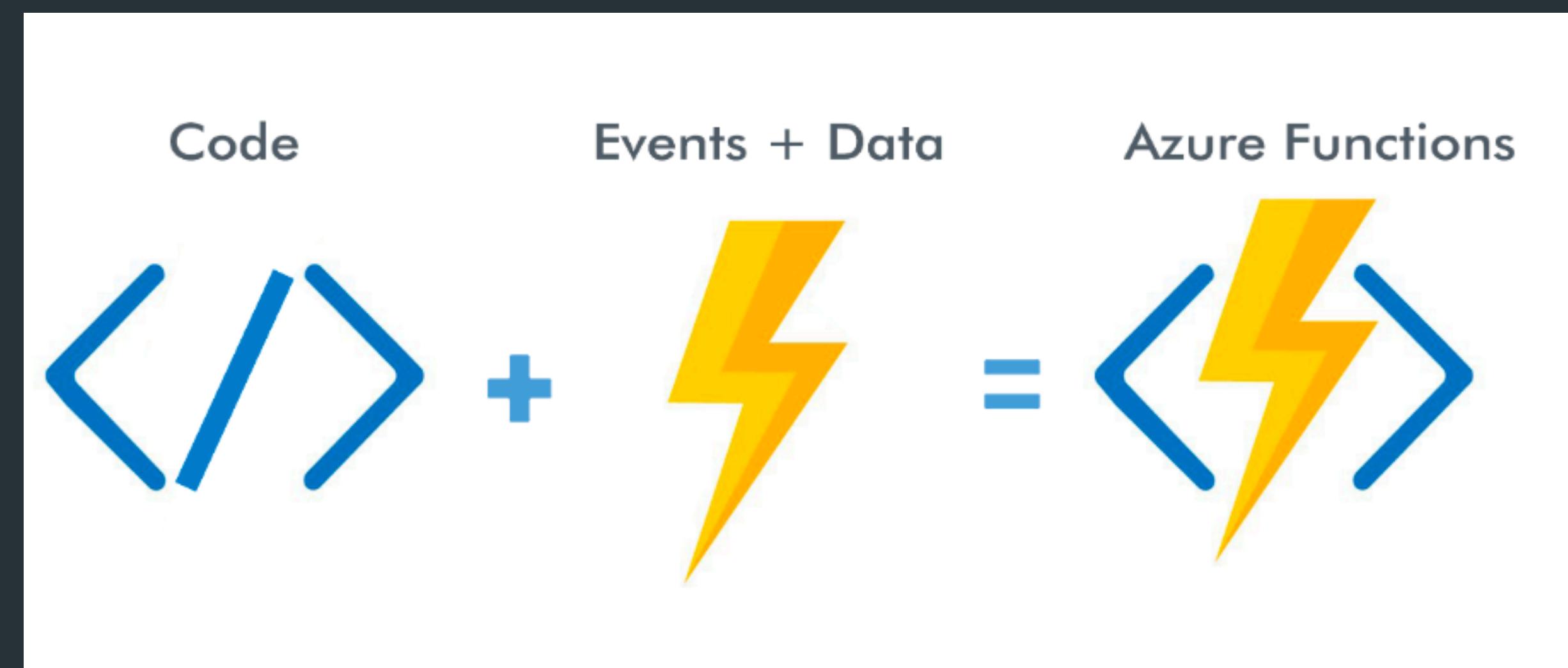
History

- 2008: Google App Engine.
- 2010: Amazon Web Services (AWS).



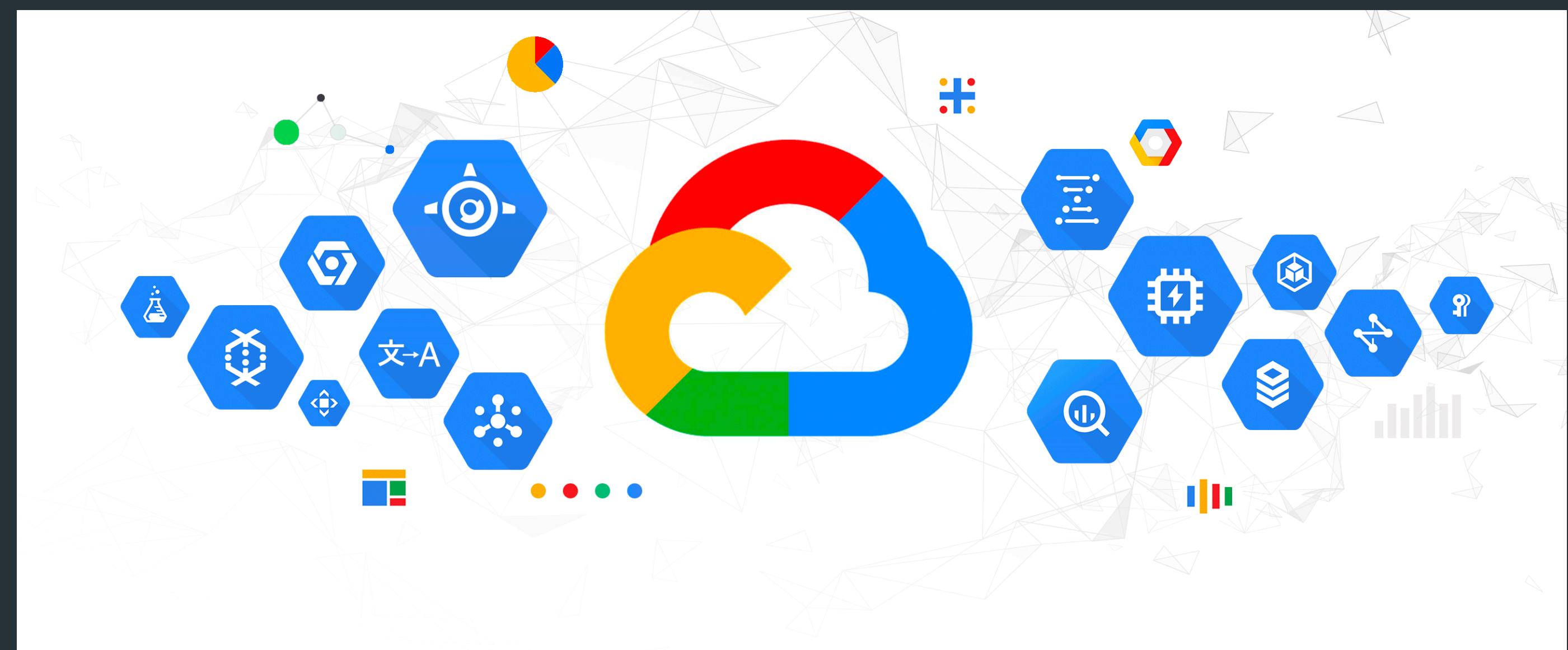
History

- 2008: Google App Engine.
- 2010: Amazon Web Services (AWS).
- 2014: Microsoft Azure launches Functions.



History

- 2008: Google App Engine.
- 2010: Amazon Web Services (AWS).
- 2014: Microsoft Azure launches Functions.
- 2015: Google Cloud Platform (GCP).



History

- 2008: Google App Engine.
- 2010: Amazon Web Services (AWS).
- 2014: Microsoft Azure launches Functions.
- 2015: Google Cloud Platform (GCP).
- 2016: IBM Cloud Functions.



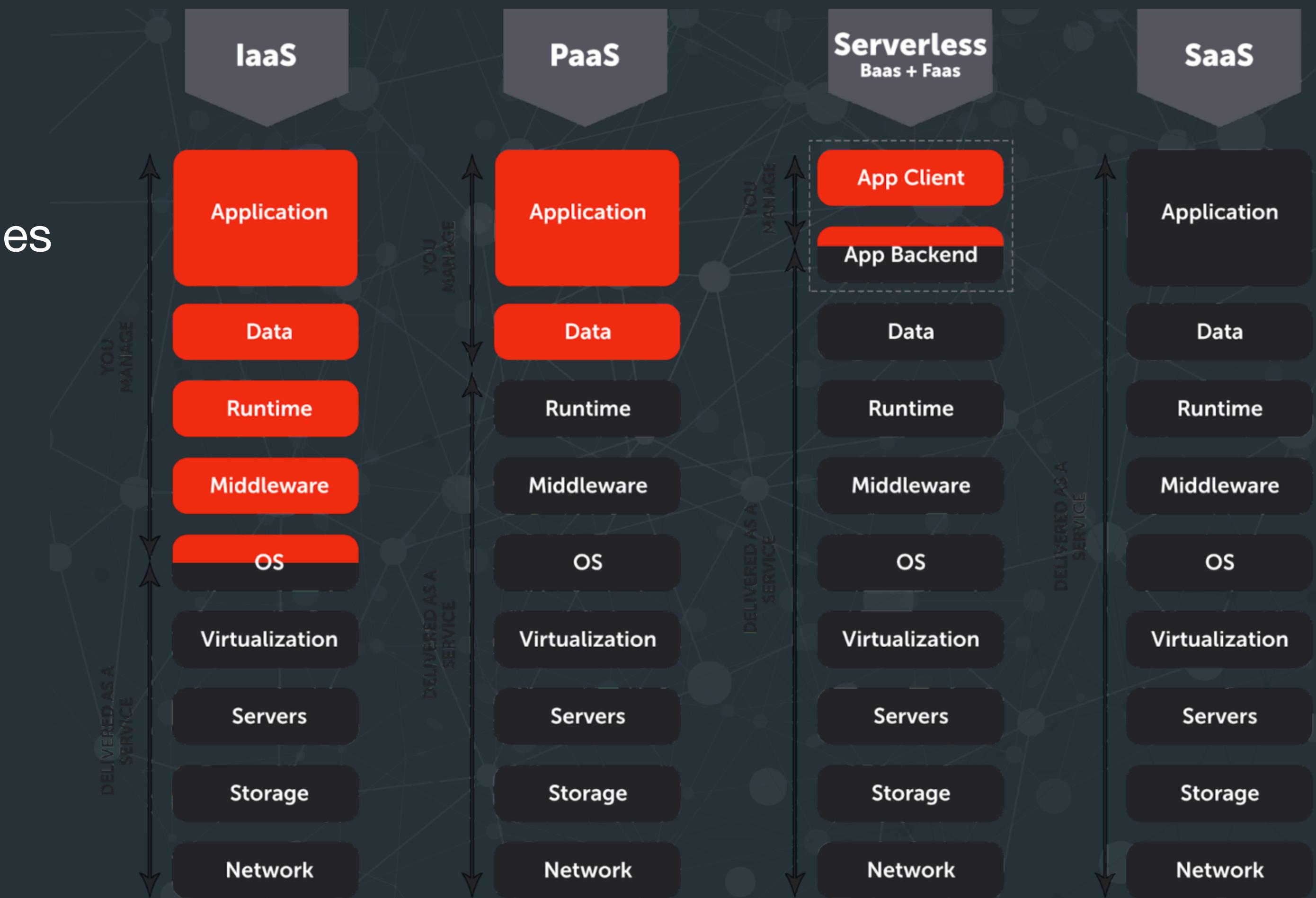
Benefits

- It offers a number of benefits over traditional server-based computing, including:
 - Reduced costs
 - Increased scalability
 - Improved agility
- Serverless computing is a good fit for a variety of applications, including:
 - Backend services
 - Event-driven applications
 - Microservices



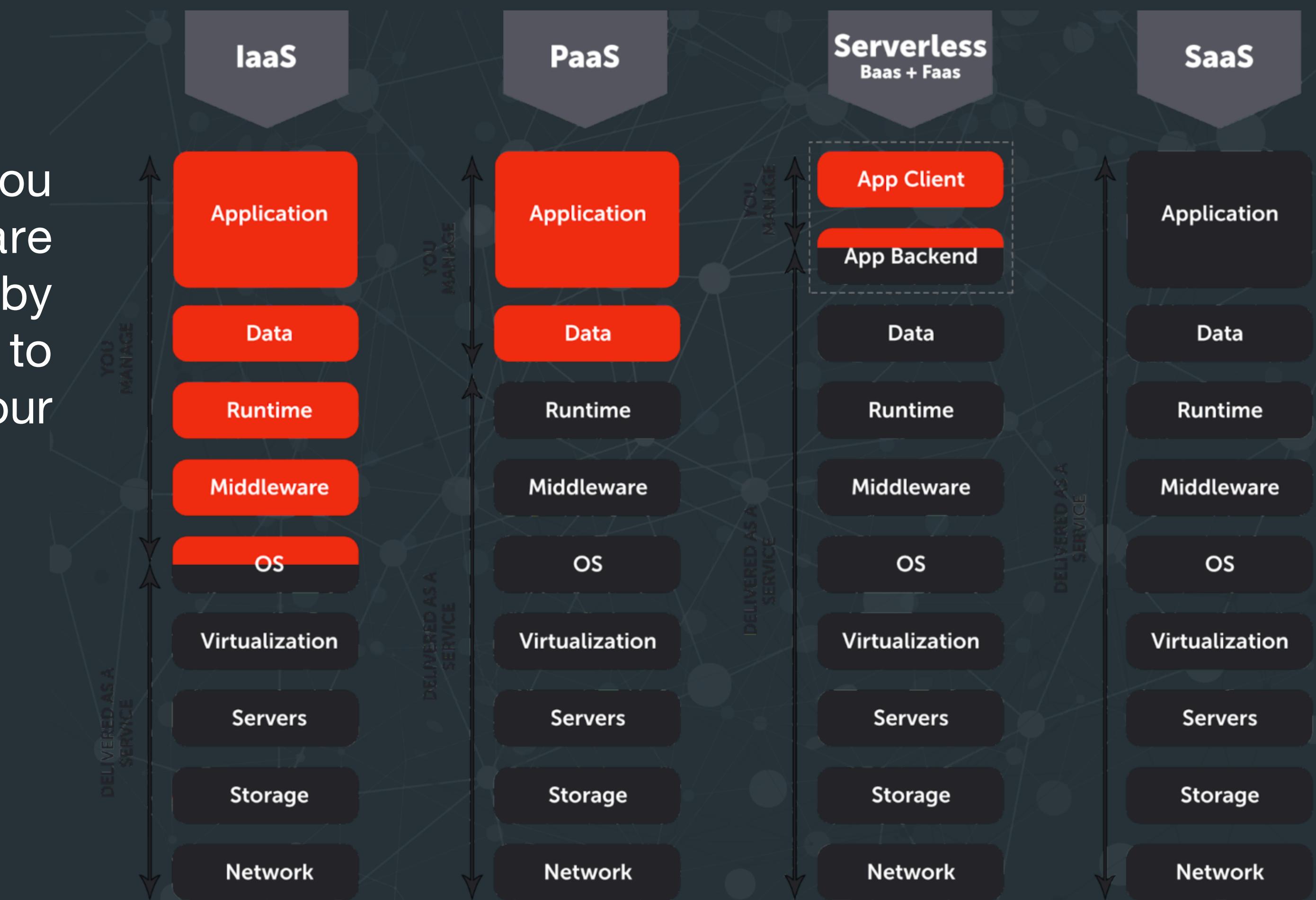
Cloud Computing Models

- Infrastructure as a Service (IaaS) provides the basic building blocks for cloud IT.



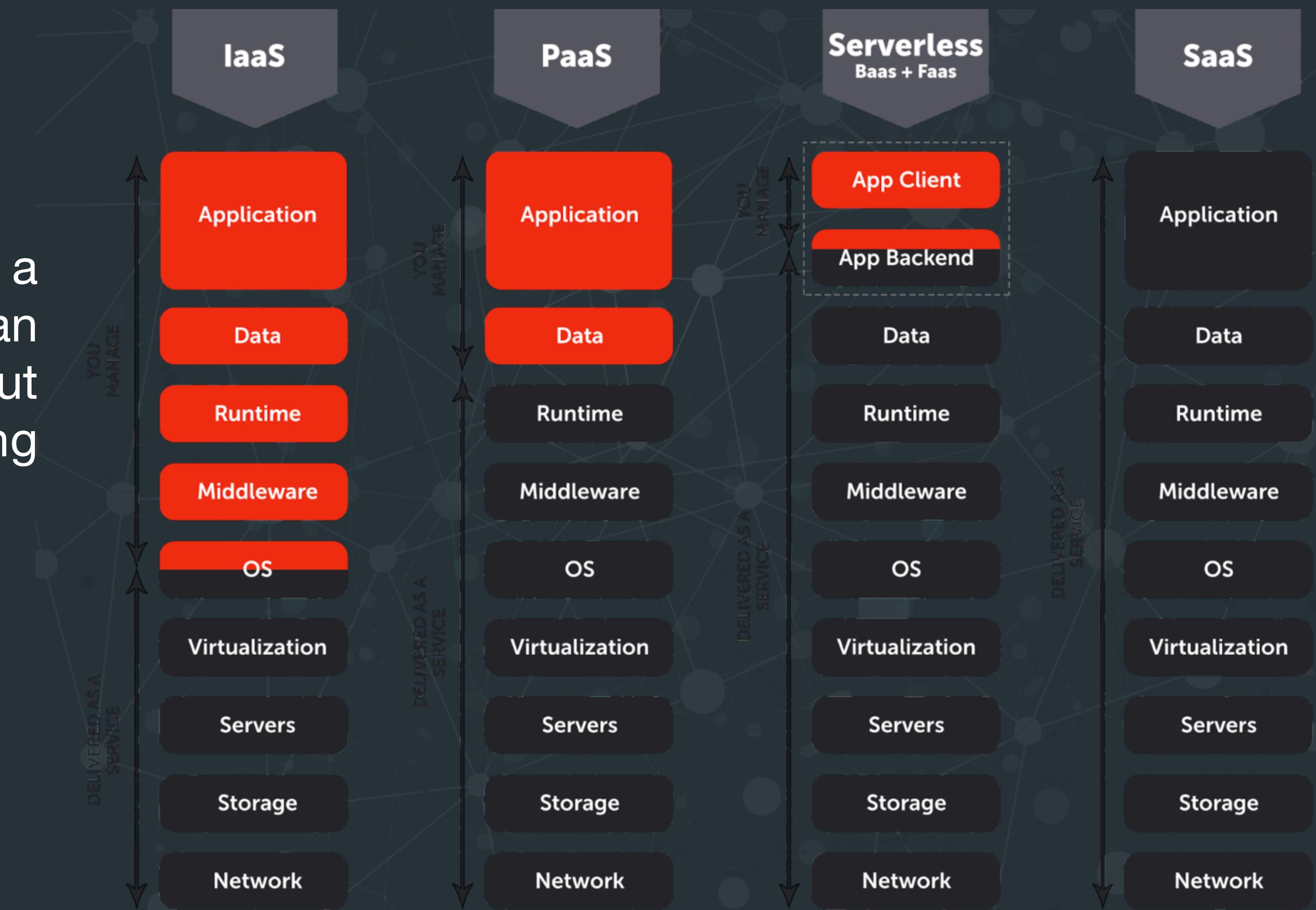
Cloud Computing Models

- Infrastructure as a Service (IaaS).
- Software as a Service (SaaS) provides you with access to a complete software application that is hosted and managed by the cloud provider. You don't need to install or maintain any software on your own computers.



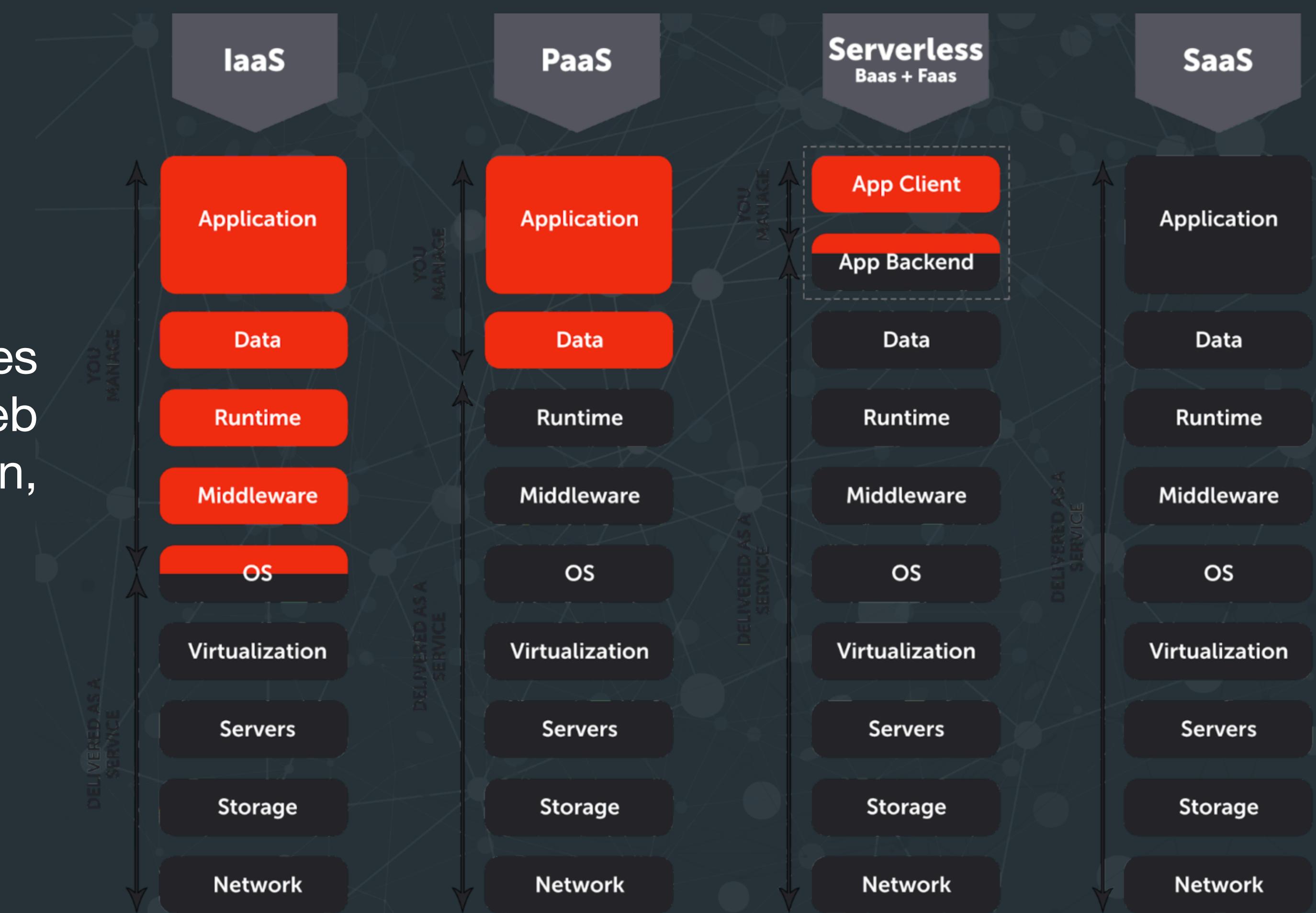
Cloud Computing Models

- Infrastructure as a Service (IaaS).
- Software as a Service (SaaS).
- Platform as a Service (PaaS) - provides a development environment where you can build, test, and deploy applications without having to worry about the underlying infrastructure.



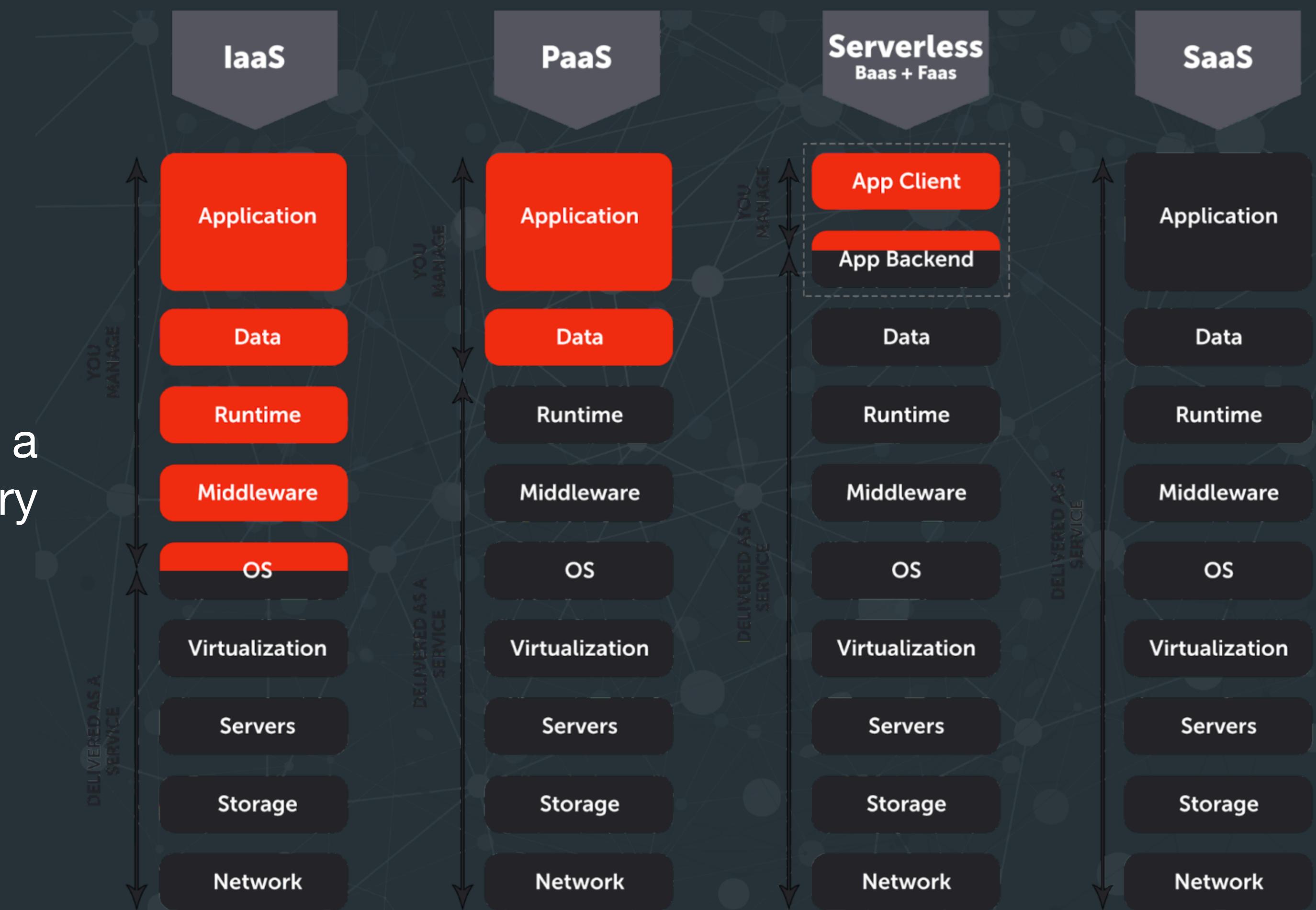
Cloud Computing Models

- Infrastructure as a Service (IaaS).
- Software as a Service (SaaS).
- Platform as a Service (PaaS).
- Backend as a Service (BaaS) provides backend services for mobile and web applications, such as user authentication, push notifications, and data storage.



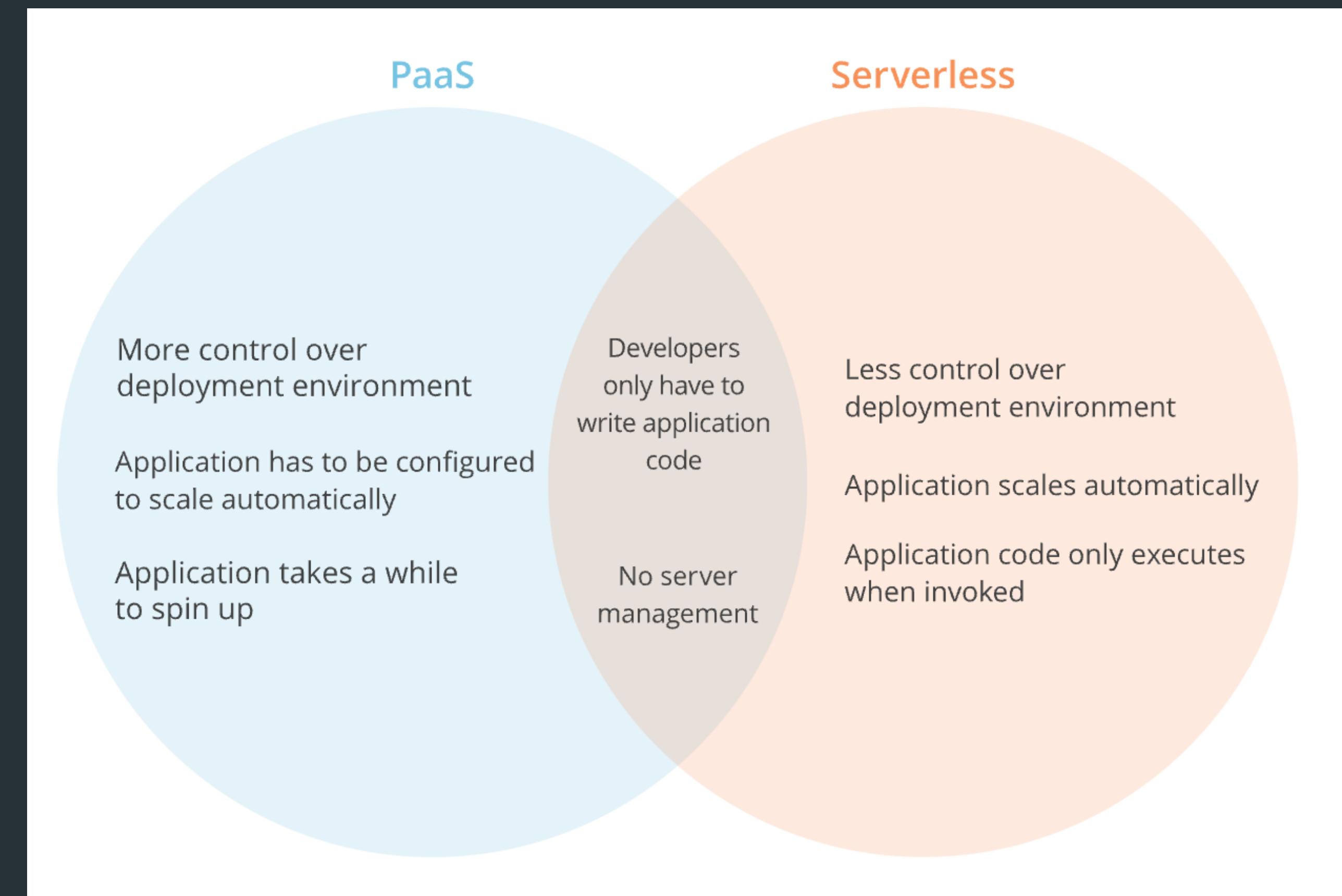
Cloud Computing Models

- Infrastructure as a Service (IaaS).
- Software as a Service (SaaS).
- Platform as a Service (PaaS).
- Backend as a Service (BaaS).
- Function as a Service (FaaS) provides a way to run code without having to worry about managing servers or infrastructure.



Types of Serverless Computing

- Function-as-a-Service (FaaS)
- Backend-as-a-Service (BaaS)
- Platform-as-a-Service (PaaS)



Function-as-a-Service (FaaS)

- The most popular type of serverless computing.
- Allows developers to run code in response to events, such as HTTP requests, database changes, or file uploads.
- Ideal for event-driven applications and for applications that need to be highly scalable and cost-effective.
- Some popular providers include AWS Lambda, Azure Functions, and Google Cloud Functions.



Backend-as-a-Service (BaaS)

- Provides developers with a backend infrastructure, such as databases, storage, and APIs.
- Ideal for developers who want to focus on developing their applications without having to worry about the underlying infrastructure.
- Some popular providers include AWS Amplify, Azure App Service, and Google Cloud Platform App Engine.



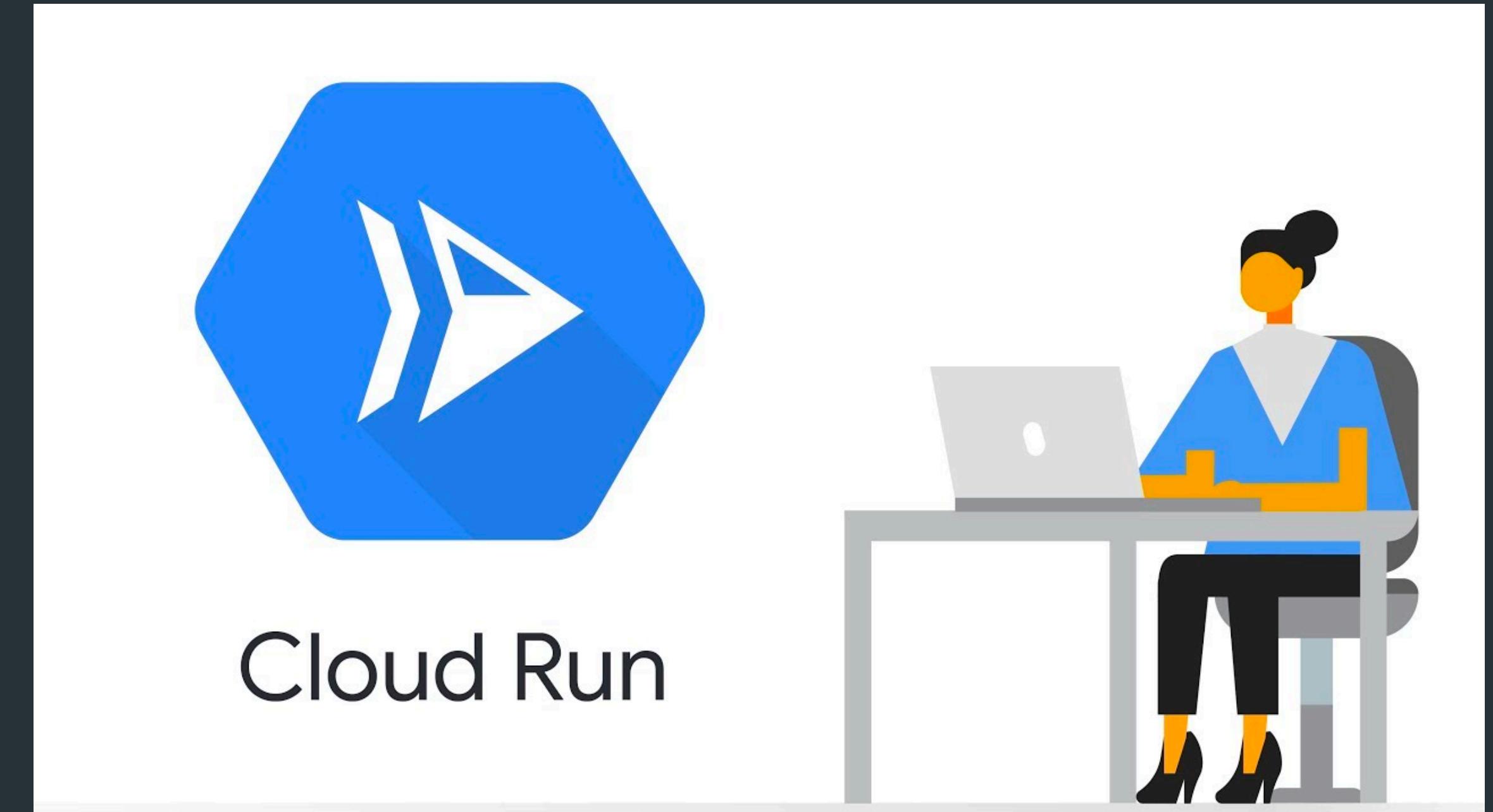
Platform-as-a-Service (PaaS)

- Provides developers with a complete development environment, including a programming language, a runtime environment, and a debugger.
- Ideal for developers who want to quickly and easily develop and deploy applications.
- Some popular providers include AWS Elastic Beanstalk, Azure App Service, and Google Cloud Platform App Engine.



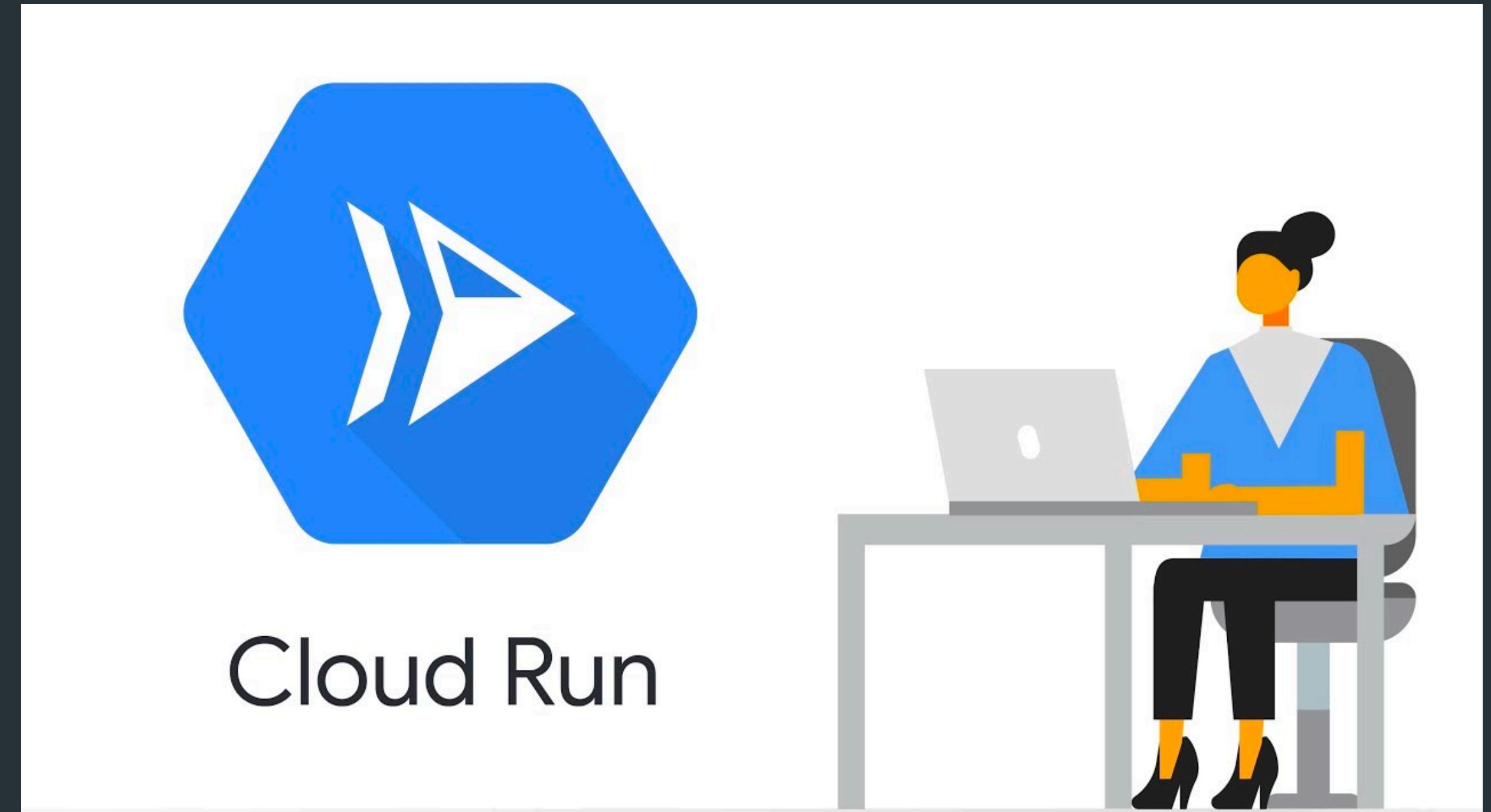
Building Serverless Applications With Google Cloud Run

- A serverless compute platform.
- Allows you to run stateless containers.
- Automatically scales your containers based on demand.



Getting Started

- To get started with Google Cloud Run, you will need to:
 - Create a Google Cloud Platform (GCP) project.
 - Enable the Cloud Run API.
 - Create a Dockerfile.
 - Build your container image.
 - Deploy your container image to Cloud Run.



Create a Google Cloud Platform (GCP) project.

The screenshot shows the 'New Project' creation interface in the Google Cloud console. On the left, there's a sidebar with a 'New Project' section containing fields for 'Project name*' (set to 'My Project 53625') and 'Location*', which is set to 'No organization'. Below these are 'CREATE' and 'CANCEL' buttons. On the right, a 'Tutorial' card titled 'Step 1 of 2' provides instructions for creating a project, mentioning the 'Manage resources' page and the 'Select organization' dropdown. It also includes links to 'Go to Manage Resources' and 'NEXT' buttons.

New Project

Project name *

My Project 53625

Project ID: unique-provider-386906.
It cannot be changed later. [EDIT](#)

Location *

No organization [BROWSE](#)

Parent organization or folder

[CREATE](#) [CANCEL](#)

LEARN Tutorial

Step 1 of 2

Create a project

To create a new project, do the following:

1. Go to the **Manage resources** page in the Google Cloud console.
[Go to Manage Resources](#)
2. On the **Select organization** drop-down list at the top of the page, select the organization resource in which you want to create a project. If you are a free trial user, skip this step, as this list does not

[PREVIOUS](#) [NEXT](#)

Cloud Run API

The screenshot shows a web browser window with the title "API Cloud Run API – APIs & Services". The URL in the address bar is "console.cloud.google.com/apis/library/run.googleapis.com?project=ubbclasses". The page displays the "Product details" for the "Cloud Run API". The API is identified by a blue double-right-pointing arrow icon and the text "Cloud Run API" and "Google Enterprise API". A description below states "Serverless agility for containerized apps". Two buttons are present: a blue "ENABLE" button and a white "TRY THIS API" button with a blue outline. At the bottom, there are three navigation links: "OVERVIEW" (underlined), "DOCUMENTATION", and "RELATED PRODUCTS". The main content area is titled "Overview" and contains the text "Run stateless HTTP containers on a fully managed environment."

Dockerfile

```
FROM python:3.7-alpine  
WORKDIR /app  
COPY requirements.txt ./  
RUN pip install -r requirements.txt  
COPY app.py ./  
CMD ["python", "app.py"]
```

Container Image

```
docker build -t my-app .
```

Container Image

```
docker build -t my-app .
```

Deploy the Image

```
gcloud run deploy my-app --image=my-app
```

Container Image

```
docker build -t my-app .
```

Deploy the Image

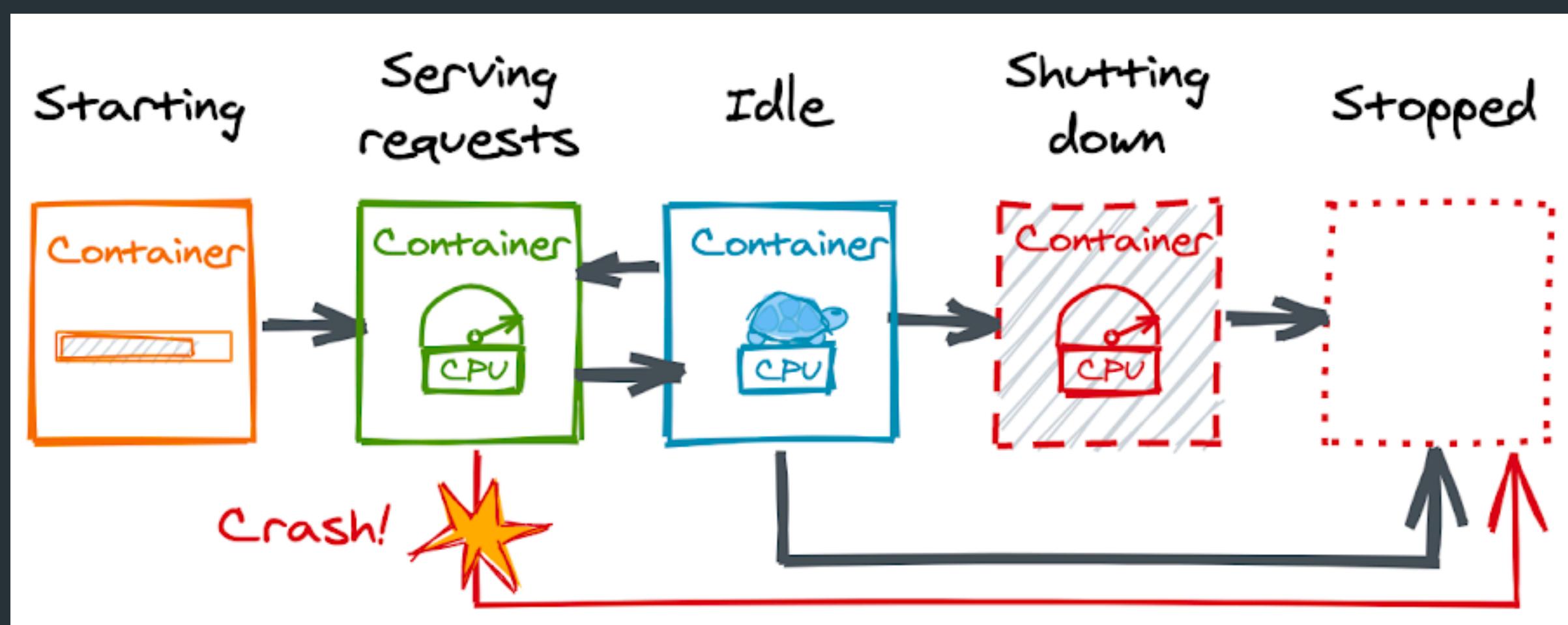
```
gcloud run deploy my-app --image=my-app
```

Access the Image

<https://my-app.cloud.run>

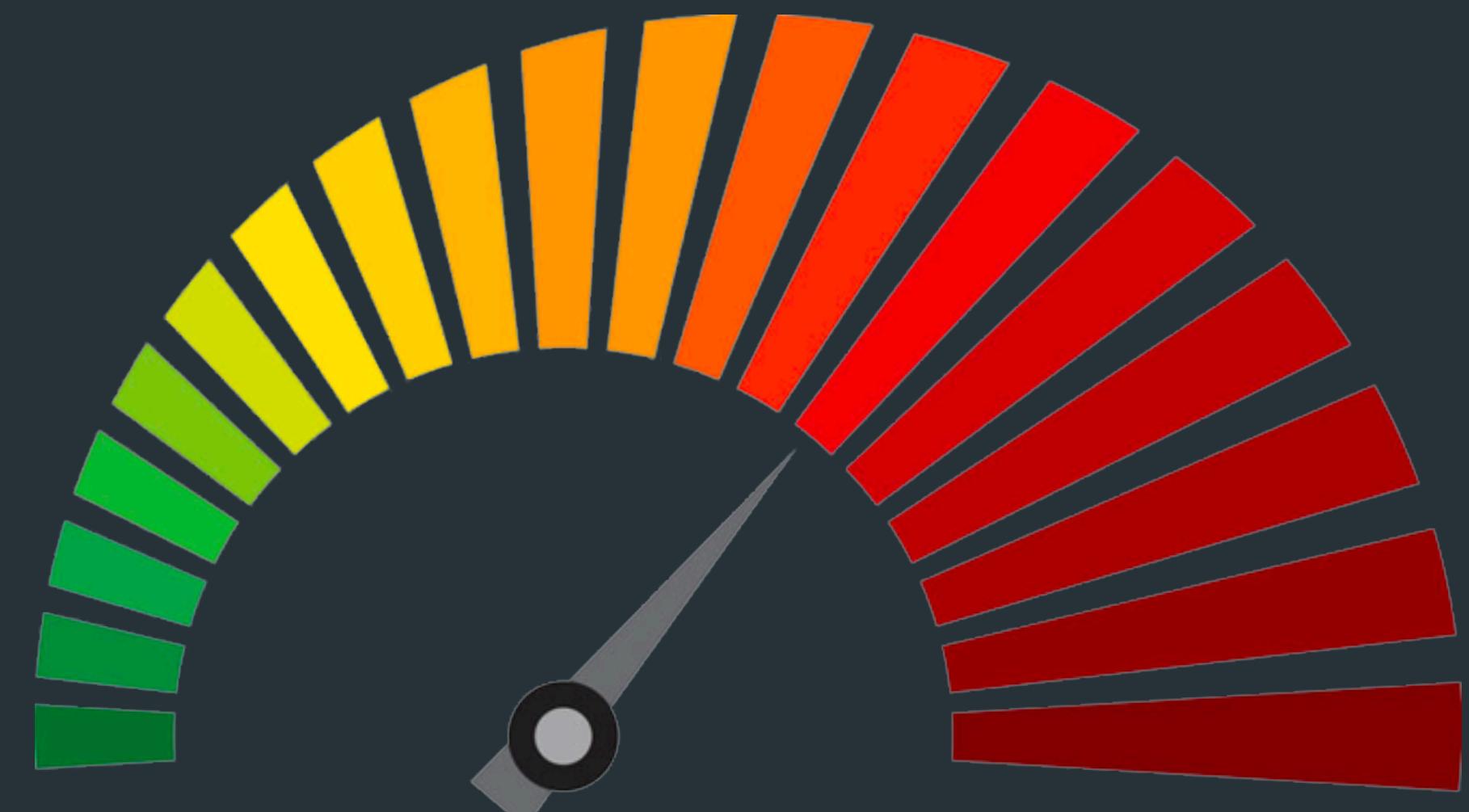
Container Lifecycle

- Creation: The container is created from a container image.
- Start: The container is started and begins listening for requests.
- Run: The container processes requests until it is terminated.
- Stop: The container is stopped and no longer listens for requests.
- Termination: The container is terminated and its resources are released.



What is CPU Throttling?

- CPU throttling is a feature of Google Cloud Run.
- CPU throttling allows you to control how much CPU your container can use.
- By default, Cloud Run will throttle your container's CPU usage to 50%.



Reasons to Throttling

- Reduce cost.
- Improve performance.



How to Use CPU Throttling

runtime: python37

env:

PORT: 8080

handlers:

- url: /.*

script: main.py

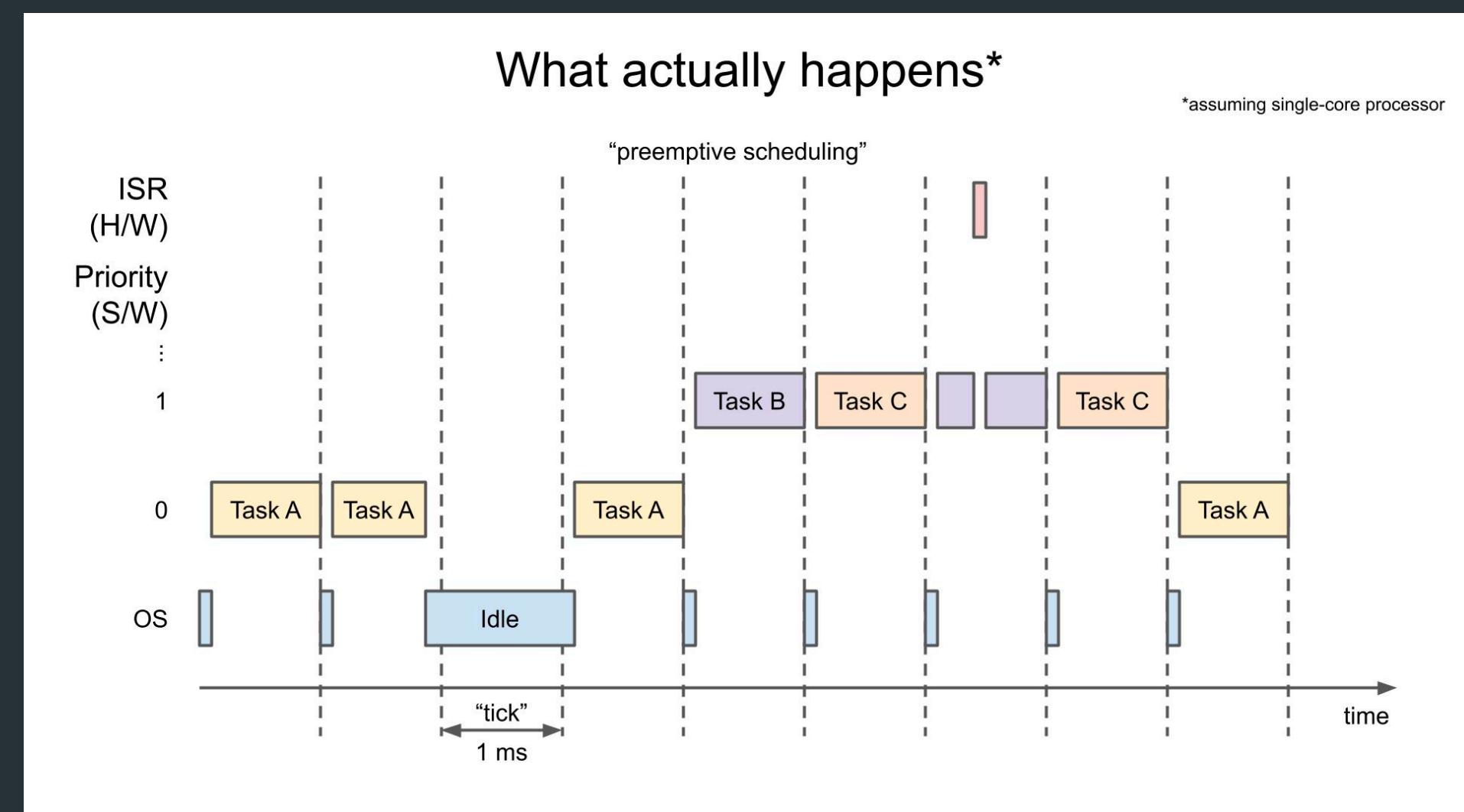
resources:

cpu: 25



Task Scheduling

- You can schedule tasks in Cloud Run using the following methods:
 - Cron jobs: Cron jobs allow you to run tasks on a recurring schedule.
 - Event-driven tasks: Event-driven tasks allow you to run tasks in response to events.

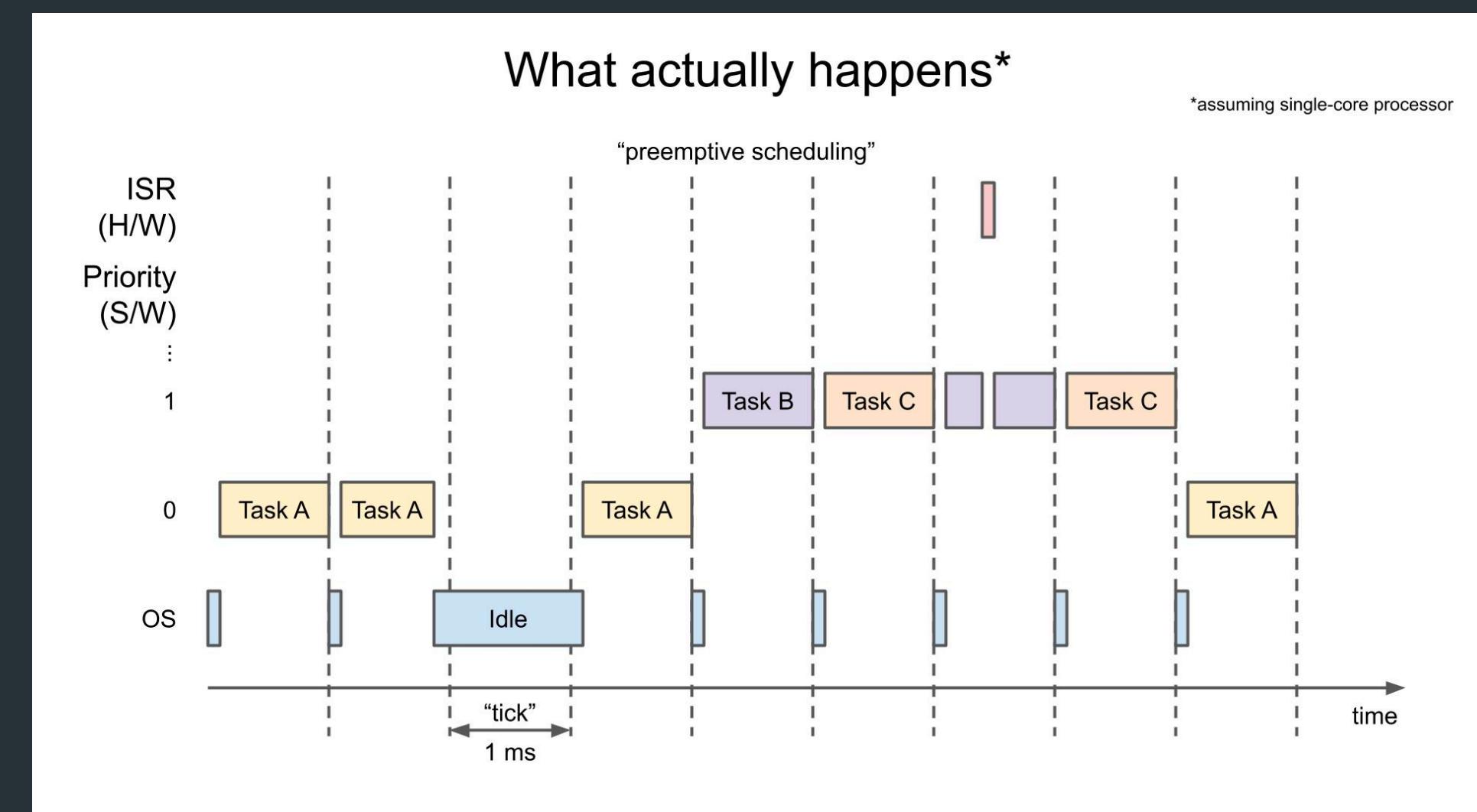


Task Scheduling and Throttling

- Throttle the CPU usage of your containers in Cloud Run using the following methods:
 - CPU quota: CPU quota allows you to specify the maximum amount of CPU that a container can use.
 - CPU requests: CPU requests allow you to specify the minimum amount of CPU that a container needs.

cron:

- description: "Run my task every day at 10am"
- schedule: "0 10 * * *"
- command: "gcloud run invoke my-task"

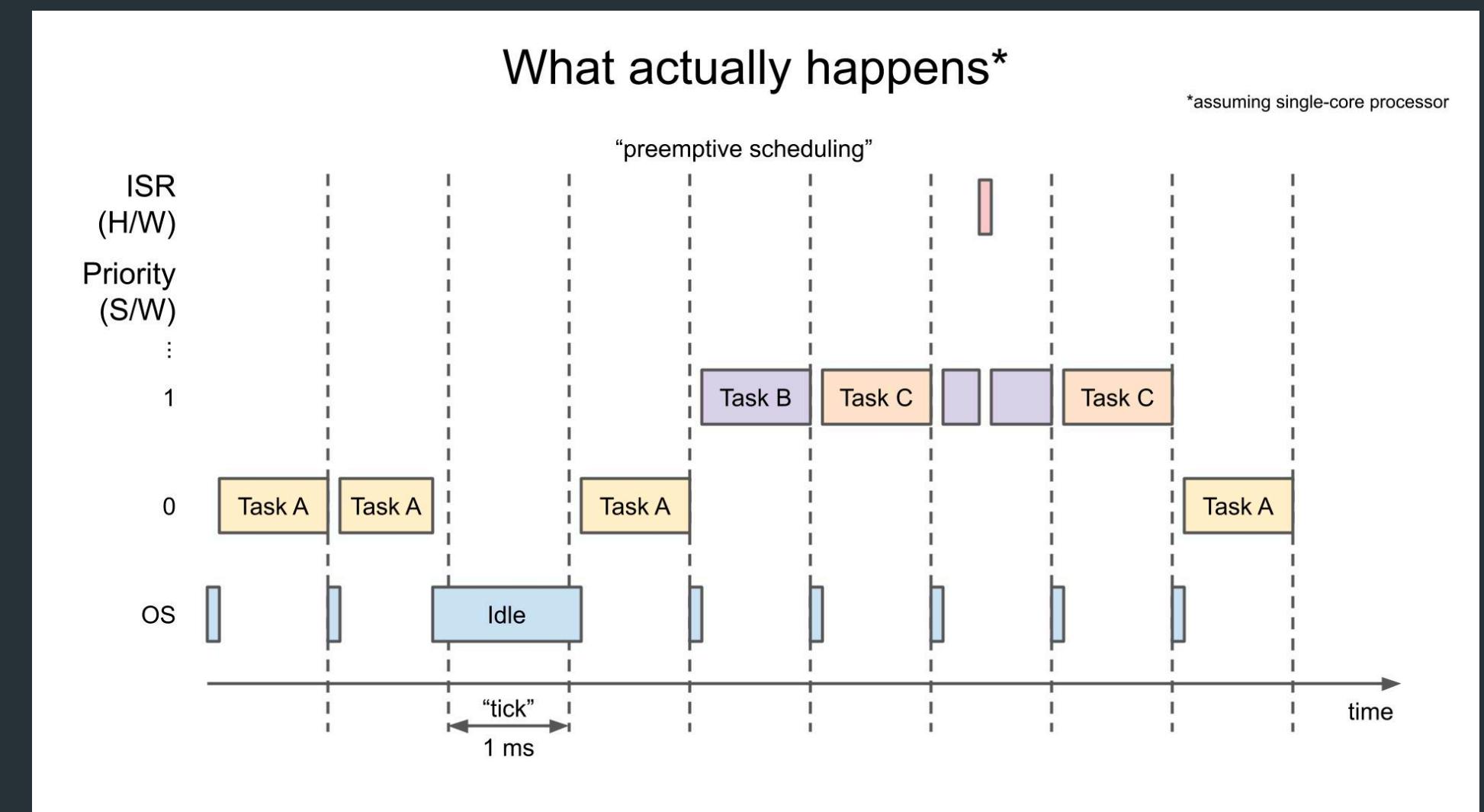


Task Scheduling and Throttling

- Throttle the CPU usage of your containers in Cloud Run using the following methods:
 - CPU quota: CPU quota allows you to specify the maximum amount of CPU that a container can use.
 - CPU requests: CPU requests allow you to specify the minimum amount of CPU that a container needs.

cron:

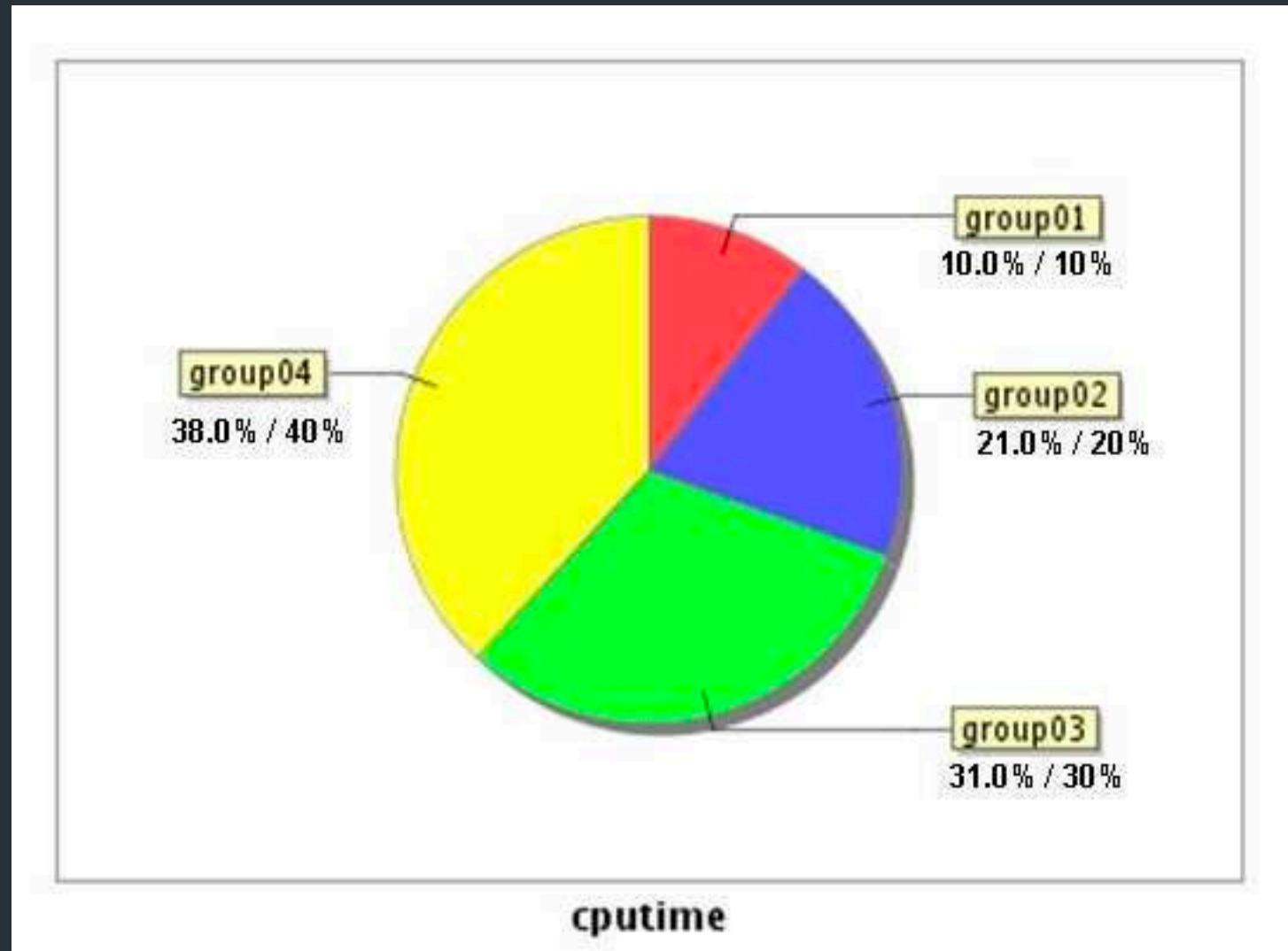
- description: "Run my task every day at 10am"
- schedule: "0 10 * * *"
- command: "gcloud run invoke my-task"



CPU Quota

```
gcloud run set-cpu-quota REGION SERVICE_NAME CPU_QUOTA
```

- Where:
 - REGION is the region where the service is running.
 - SERVICE_NAME is the name of the service.
 - CPU_QUOTA is the number of CPU cores that you want to allocate to the service.

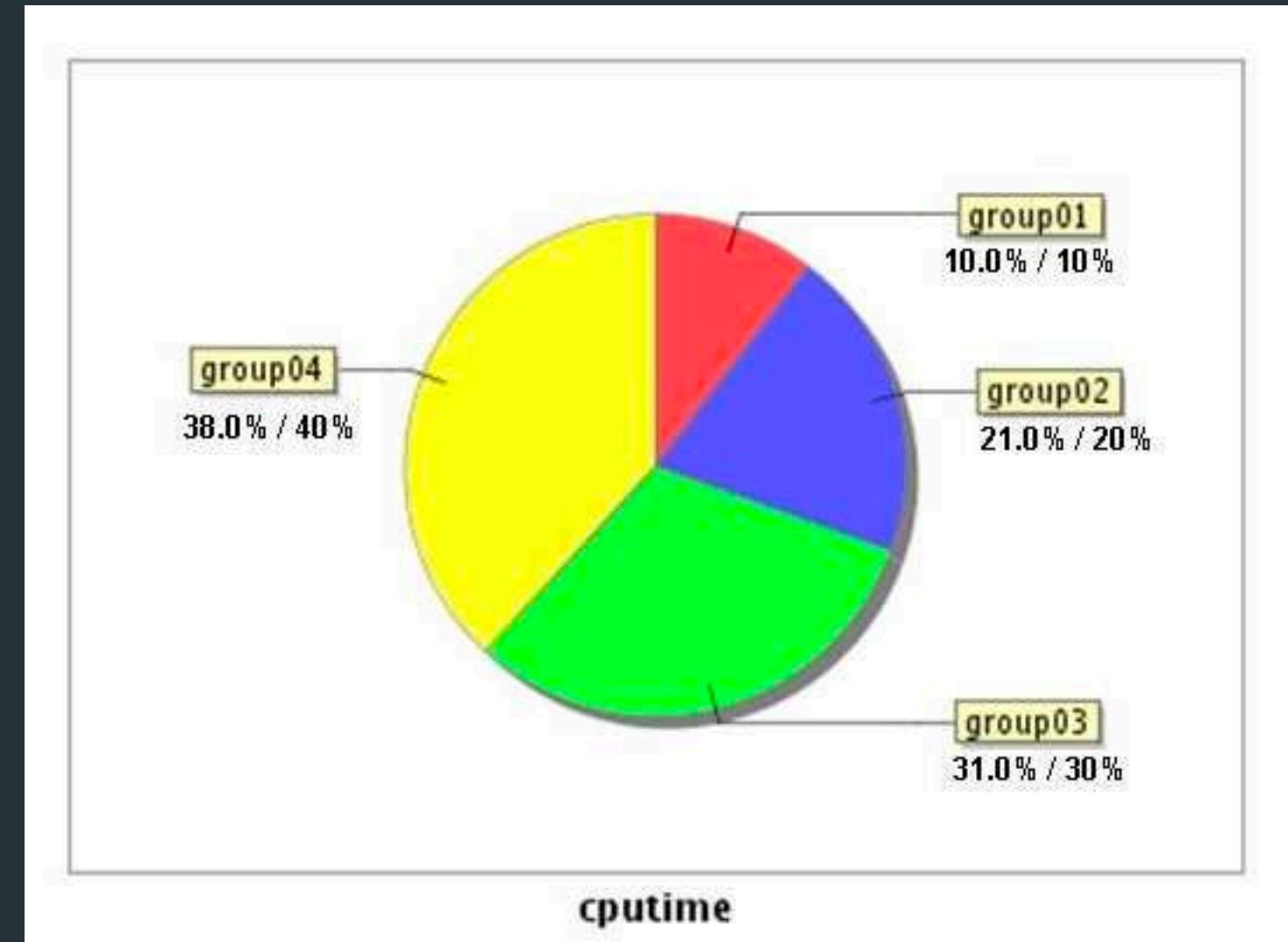


CPU Quota

```
gcloud run set-cpu-quota REGION SERVICE_NAME CPU_QUOTA
```

- Where:
 - REGION is the region where the service is running.
 - SERVICE_NAME is the name of the service.
 - CPU_QUOTA is the number of CPU cores that you want to allocate to the service.

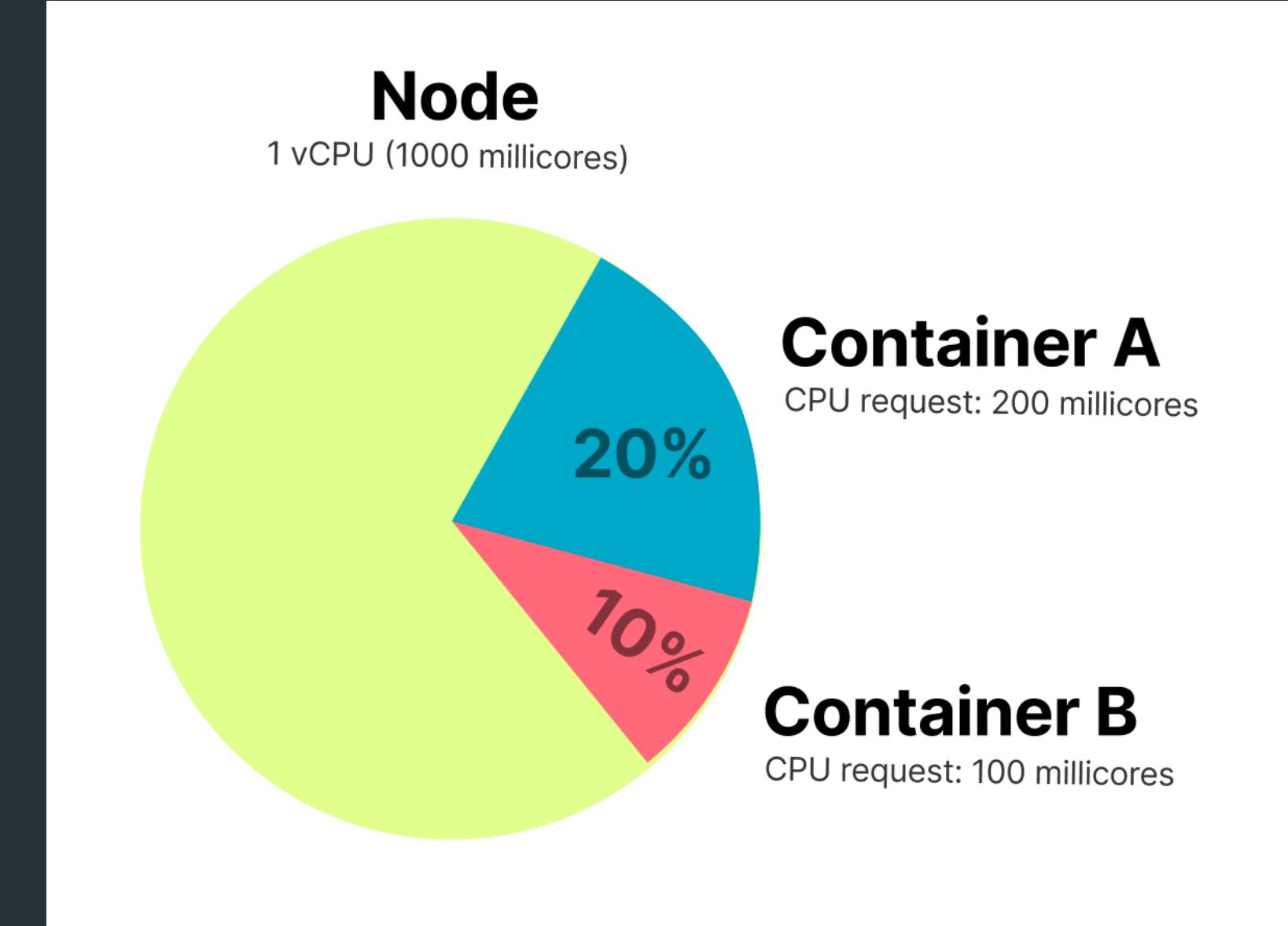
```
gcloud run set-cpu-quota us-central1 my-service 1
```



CPU Requests

```
gcloud run set-cpu-request REGION SERVICE_NAME CONTAINER_NAME CPU_REQUEST
```

- Where:
 - REGION is the region where the service is running.
 - SERVICE_NAME is the name of the service.
 - CONTAINER_NAME is the name of the container.
 - CPU_REQUEST is the number of CPU cores that you want to request for the container.

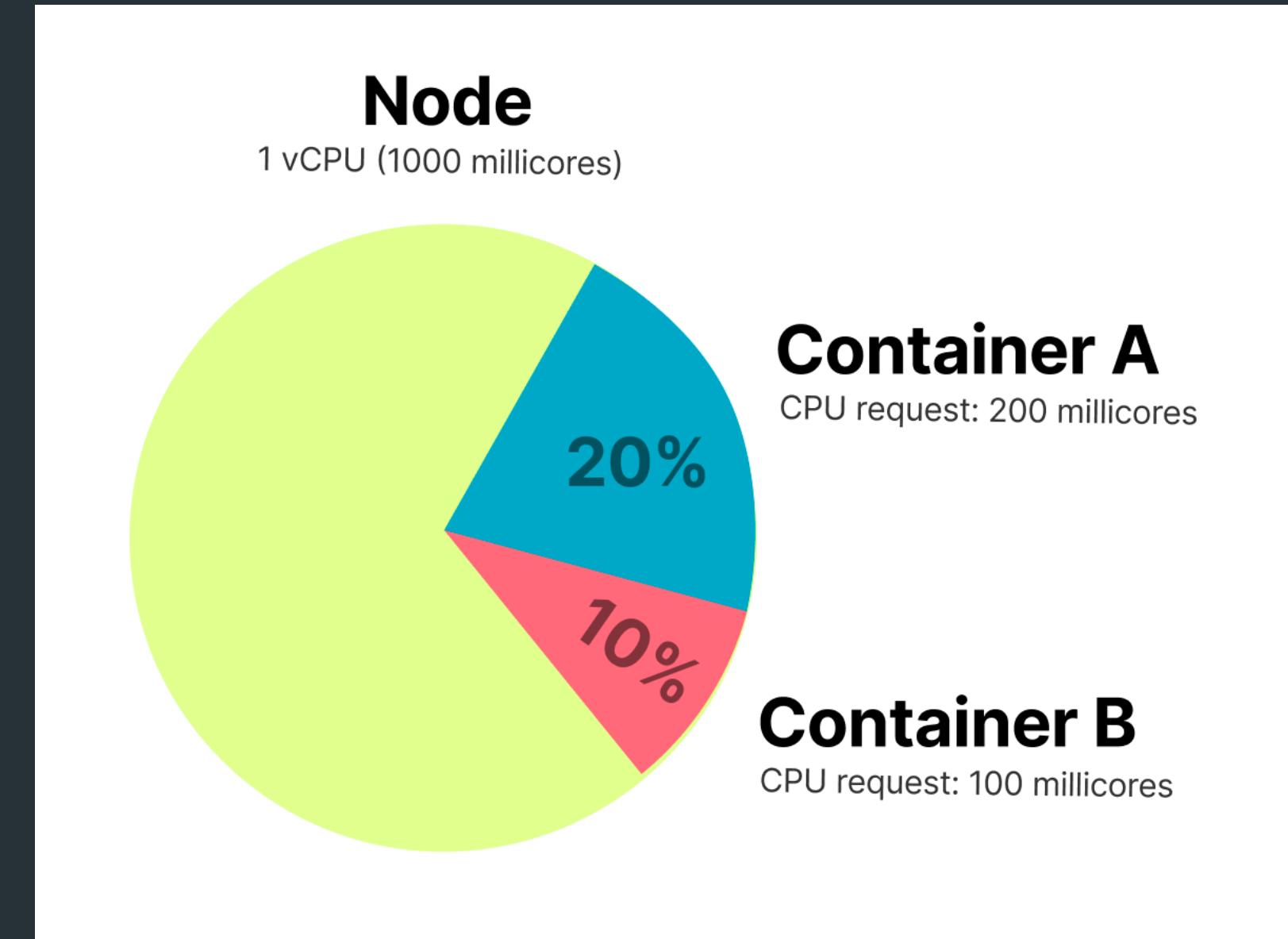


CPU Requests

```
gcloud run set-cpu-request REGION SERVICE_NAME CONTAINER_NAME CPU_REQUEST
```

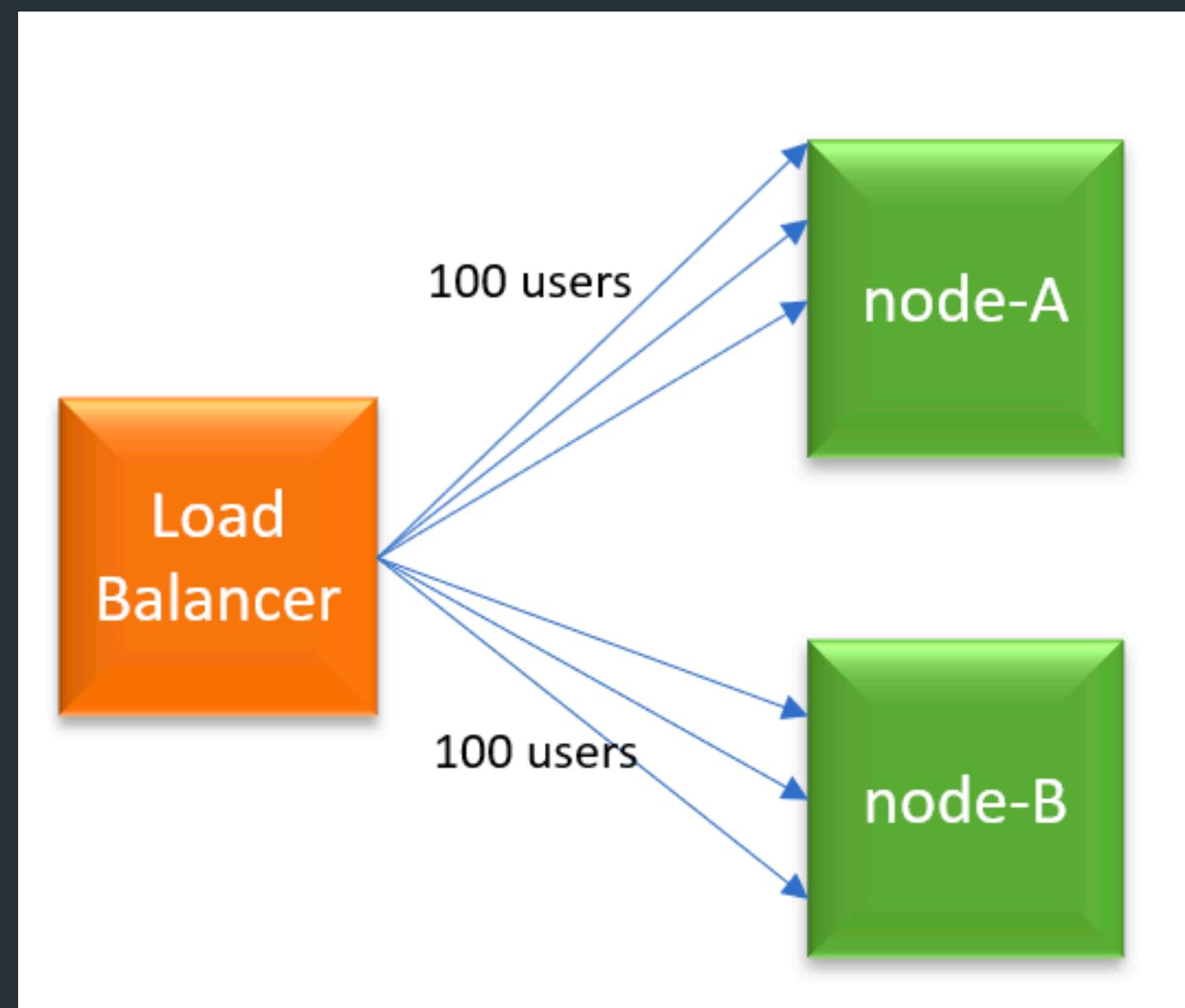
- Where:
 - REGION is the region where the service is running.
 - SERVICE_NAME is the name of the service.
 - CONTAINER_NAME is the name of the container.
 - CPU_REQUEST is the number of CPU cores that you want to request for the container.

```
gcloud run set-cpu-request us-central1 my-service my-container 0.5
```



Load Balancing

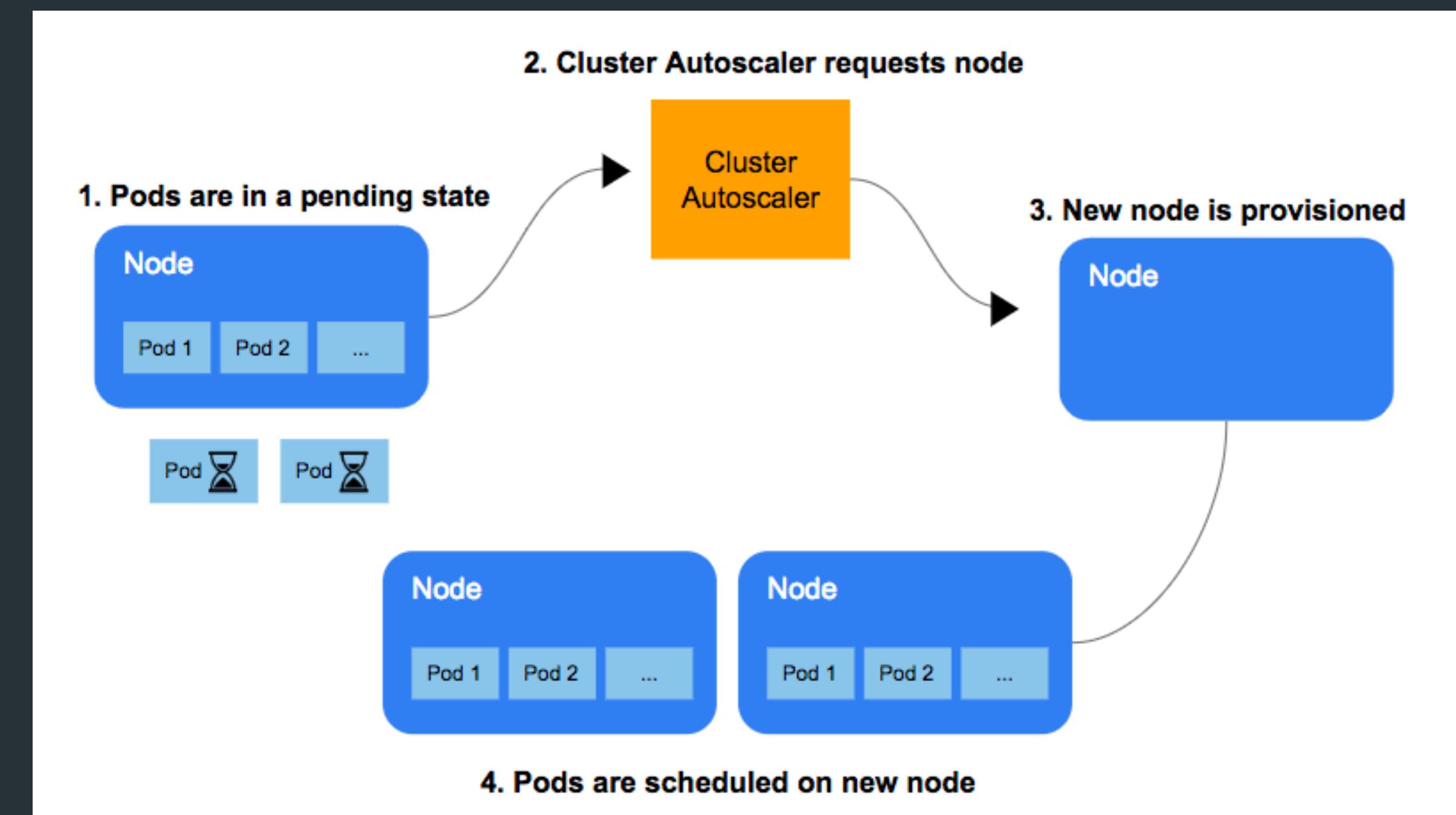
Cloud Run uses a round-robin load balancer to distribute traffic across your containers. This means that each container will receive an equal number of requests.



Autoscaler

Autoscaler automatically scales your containers up or down based on demand. The autoscaler uses a variety of factors to determine when to scale, including:

- the number of requests per second.
- the CPU usage.
- the memory usage.



Example

```
loadBalancer:  
  enabled: true  
  type: ROUND_ROBIN
```

```
autoscaler:  
  minCount: 1  
  maxCount: 10
```

Building Serverless Applications With AWS

- A serverless compute platform.
- Allows you to run stateless containers.
- Automatically scales your containers based on demand.

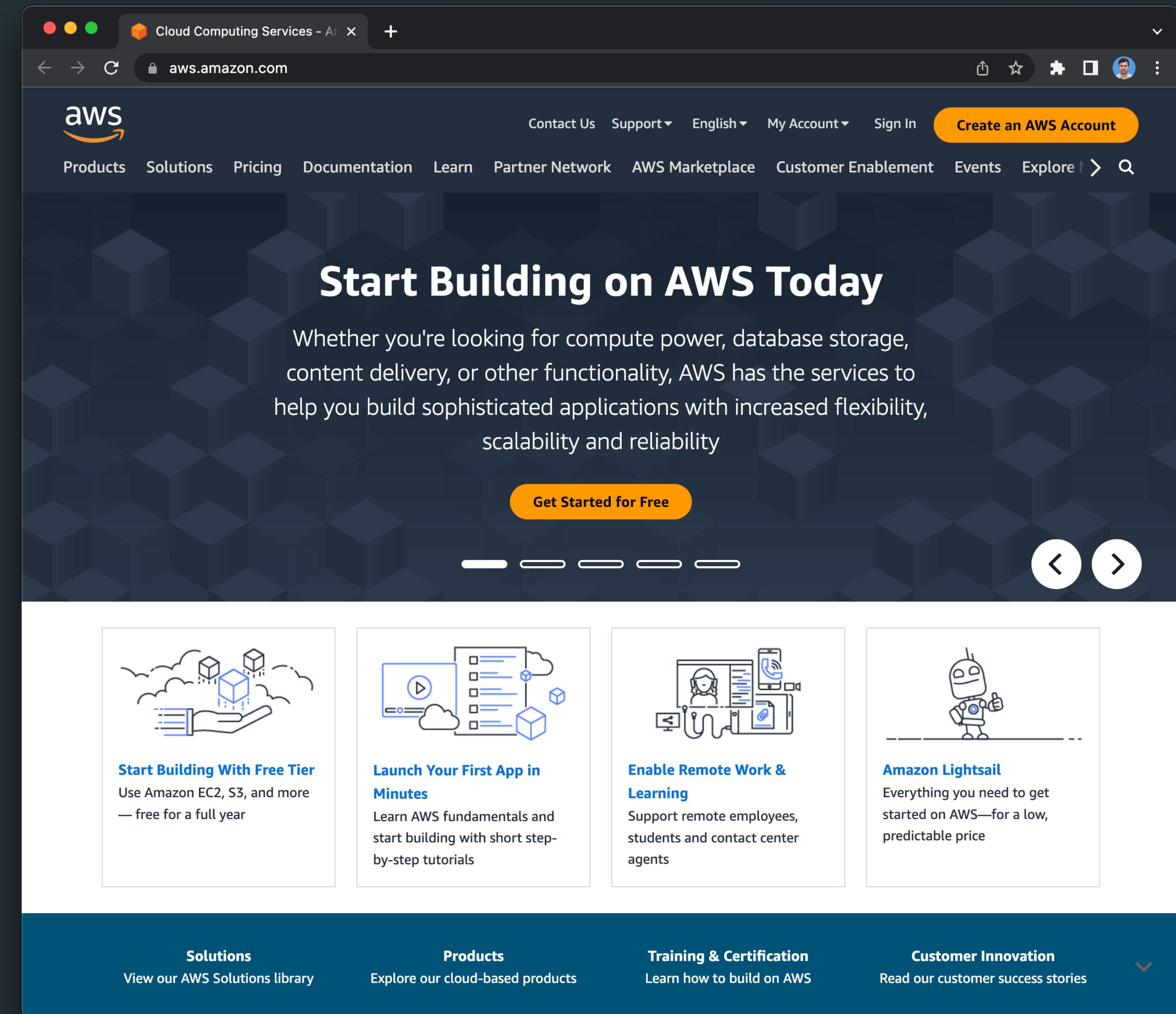


AWS Serverless Services

- **AWS Lambda:** AWS Lambda is a service that allows you to run code without having to provision or manage servers.
- **AWS API Gateway:** AWS API Gateway is a service that allows you to create, publish, monitor, and secure RESTful APIs.
- **AWS DynamoDB:** AWS DynamoDB is a fully managed NoSQL database service.
- **AWS CloudFormation:** AWS CloudFormation is a service that allows you to create and manage AWS resources as a single unit.



Create an AWS account



The screenshot shows the AWS homepage with a dark blue header. At the top right, there are links for "Contact Us", "Support", "English", "My Account", "Sign In", and a prominent orange "Create an AWS Account" button. Below the header, the navigation bar includes links for "Products", "Solutions", "Pricing", "Documentation", "Learn", "Partner Network", "AWS Marketplace", "Customer Enablement", "Events", "Explore", and a search icon. The main content area features a large banner with the heading "Start Building on AWS Today". The banner text explains that AWS offers services for compute power, database storage, content delivery, and other functionality to build sophisticated applications with increased flexibility, scalability, and reliability. A central call-to-action button says "Get Started for Free". Below the banner, there are four cards with icons and descriptions: "Start Building With Free Tier" (using Amazon EC2, S3, and more), "Launch Your First App in Minutes" (learning AWS fundamentals and building with step-by-step tutorials), "Enable Remote Work & Learning" (supporting remote employees, students, and contact center agents), and "Amazon Lightsail" (getting started on AWS for a low, predictable price). At the bottom, a teal footer bar contains links for "Solutions", "Products", "Training & Certification", and "Customer Innovation".

Cloud Computing Services - Ar X

aws.amazon.com

Contact Us Support English My Account Sign In Create an AWS Account

Products Solutions Pricing Documentation Learn Partner Network AWS Marketplace Customer Enablement Events Explore

Start Building on AWS Today

Whether you're looking for compute power, database storage, content delivery, or other functionality, AWS has the services to help you build sophisticated applications with increased flexibility, scalability and reliability

Get Started for Free

Start Building With Free Tier

Use Amazon EC2, S3, and more — free for a full year

Launch Your First App in Minutes

Learn AWS fundamentals and start building with short step-by-step tutorials

Enable Remote Work & Learning

Support remote employees, students and contact center agents

Amazon Lightsail

Everything you need to get started on AWS—for a low, predictable price

Solutions

View our AWS Solutions library

Products

Explore our cloud-based products

Training & Certification

Learn how to build on AWS

Customer Innovation

Read our customer success stories

Install the AWS CLI

If you are updating to the latest version, use the same installation method that you used in your current version. You can install the AWS CLI on macOS in the following ways.

The following steps show how to install the latest version of the AWS CLI by using the standard macOS user interface and your browser.

1. In your browser, download the macOS pkg file: <https://awscli.amazonaws.com/AWSCLIV2.pkg>
2. Run your downloaded file and follow the on-screen instructions. You can choose to install the AWS CLI in the following ways:
 - **For all users on the computer (requires sudo)**
 - You can install to any folder, or choose the recommended default folder of /usr/local/aws-cli.
 - The installer automatically creates a symlink at /usr/local/bin/aws that links to the main program in the installation folder you chose.
 - **For only the current user (doesn't require sudo)**
 - You can install to any folder to which you have write permission.
 - Due to standard user permissions, after the installer finishes, you must manually create a symlink file in your \$PATH that points to the aws and aws_completer programs by using the following commands at the command prompt. If your \$PATH includes a folder you can write to, you can run the following command without sudo if you specify that folder as the target's path. If you don't have a writable folder in your \$PATH, you must use sudo in the commands to get permissions to write to the specified target folder. The default location for a symlink is /usr/local/bin/.

```
$ sudo ln -s /folder/installed/aws-cli/aws /usr/local/bin/aws
$ sudo ln -s /folder/installed/aws-cli/aws_completer /usr/local/bin/aws
```

Note

You can view debug logs for the installation by pressing **Cmd+L** anywhere in the installer. This

Create a Lambda function

```
def hello(event, context):  
    return "Hello, " + event['name']
```

Create a deployment package

1. Create a directory for your Lambda function.
2. In the directory, create a file called `index.py` and paste your Lambda function's code into it.
3. If your Lambda function uses any dependencies, add them to the directory.
4. Zip up the directory.

```
mkdir my_function  
cd my_function  
echo "def my_function(event, context):  
    return event['message'] + '!'> index.py  
pip install numpy  
zip -r my_function.zip .
```

Create an execution role for your Lambda function

```
aws iam create-role --role-name my-lambda-role --assume-role-policy-document file://trust-policy.json
```

Create an execution role for your Lambda function

```
aws iam create-role --role-name my-lambda-role --assume-role-policy-document file://trust-policy.json
```

Attach the **AWSLambdaBasicExecutionRole** policy to the role

```
aws iam attach-policy  
  --policy-arn arn:aws:iam::aws:policy/AWSLambdaBasicExecutionRole  
  --role-name my-lambda-role
```

Create an execution role for your Lambda function

```
aws iam create-role --role-name my-lambda-role --assume-role-policy-document file://trust-policy.json
```

Attach the AWSLambdaBasicExecutionRole policy to the role

```
aws iam attach-policy  
  --policy-arn arn:aws:iam::aws:policy/AWSLambdaBasicExecutionRole  
  --role-name my-lambda-role
```

Deploy the Lambda function

```
aws lambda create-function --function-name my-lambda-function  
  --runtime python3.8  
  --role arn:aws:iam::<your-account-id>:role/my-lambda-role  
  --handler index.my_function  
  --zip-file file://my_function.zip
```

AWS API Gateway

- Visual editor
- CORS support
- Authorization and access control
- Monitoring and logging
- Deployment



Getting started with API Gateway X +

docs.aws.amazon.com/apigateway/latest/developerguide/getting-started.html

Contact Us English Create an AWS Account

AWS Documentation Amazon API Gateway Developer Guide Feedback Preferences

Amazon API Gateway

Developer Guide

- What is Amazon API Gateway?
- Prerequisites
- Getting started**
- Tutorials and workshops
- Working with REST APIs
- Working with HTTP APIs
- Working with WebSocket APIs
- API Gateway ARNs
- OpenAPI extensions
- Security
- Tagging
- API references
- Quotas and important notes
- Document history
- AWS glossary

Step 1: Create a Lambda function

You use a Lambda function for the backend of your API. Lambda runs your code only when needed and scales automatically, from a few requests per day to thousands per second.

For this example, you use the default Node.js function from the Lambda console.

To create a Lambda function

- Sign in to the Lambda console at <https://console.aws.amazon.com/lambda>.
- Choose **Create function**.
- For **Function name**, enter `my-function`.
- Choose **Create function**.

The example function returns a `200` response to clients, and the text `Hello from Lambda!`.

You can modify your Lambda function, as long as the function's response aligns with the [format that API Gateway requires](#).

The default Lambda function code should look similar to the following:

```
export const handler = async (event) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

The screenshot shows a web browser window with the URL docs.aws.amazon.com/apigateway/latest/developerguide/getting-started.html. The page is titled "Getting started with API Gateway". The left sidebar contains a navigation menu for the "Amazon API Gateway" developer guide, with the "Getting started" section currently selected. The main content area is titled "Step 2: Create an HTTP API". It explains that next, an HTTP API will be created, noting that API Gateway also supports REST APIs and WebSocket APIs. It highlights that an HTTP API is the best choice for this exercise because REST APIs support more features than HTTP APIs, and HTTP APIs are designed with minimal features to be offered at a lower price. It also mentions that WebSocket APIs maintain persistent connections with clients for full-duplex communication. Below this, a section titled "To create an HTTP API" provides a numbered list of steps:

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. Do one of the following:
 - To create your first API, for **HTTP API**, choose **Build**.
 - If you've created an API before, choose **Create API**, and then choose **Build** for **HTTP API**.
3. For **Integrations**, choose **Add integration**.
4. Choose **Lambda**.
5. For **Lambda function**, enter `my-function`.
6. For **API name**, enter `my-http-api`.
7. Choose **Next**.
8. Review the *route* that API Gateway creates for you, and then choose **Next**.
9. Review the *stage* that API Gateway creates for you, and then choose **Next**.
10. Choose **Create**.

At the bottom, a note states: "Now you've created an HTTP API with a Lambda integration that's ready to receive requests from clients."

Getting started with API Gateway

docs.aws.amazon.com/apigateway/latest/developerguide/getting-started.html

AWS Search in this guide Contact Us English Create an AWS Account

AWS Documentation Amazon API Gateway Developer Guide Feedback Preferences

Amazon API Gateway

Developer Guide

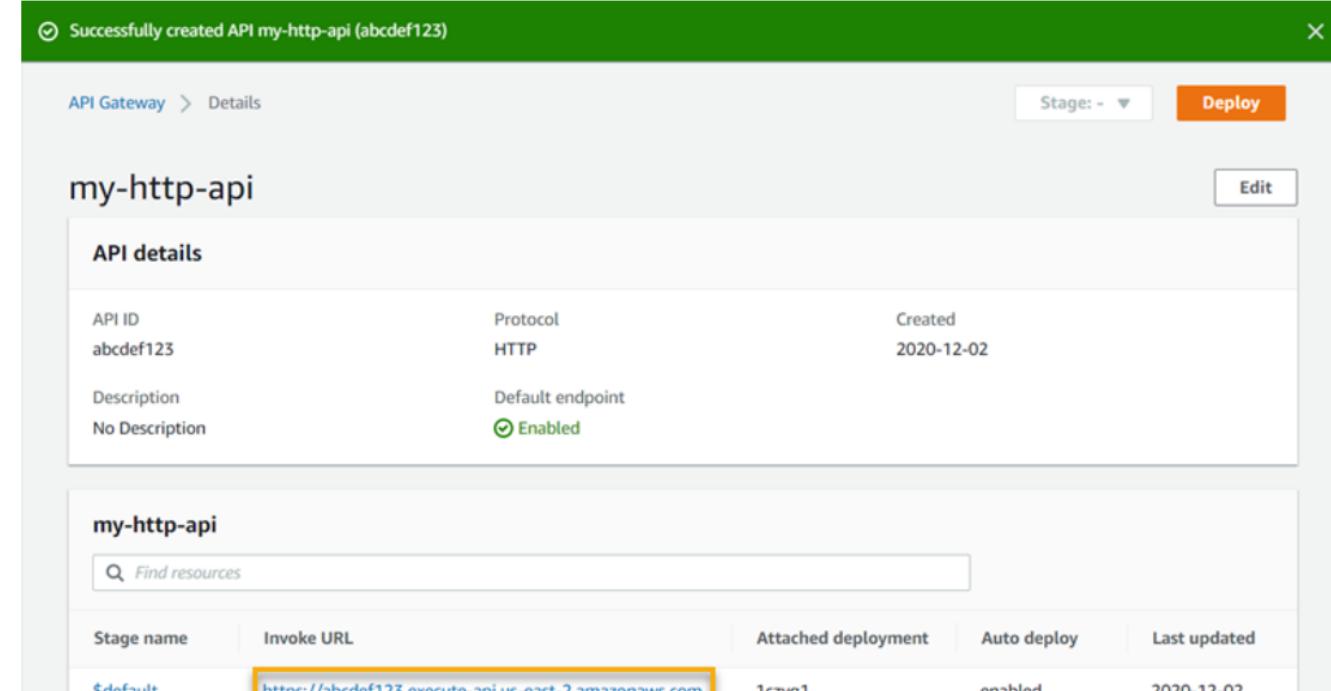
- ▶ What is Amazon API Gateway?
- ▶ Prerequisites
- Getting started**
- ▶ Tutorials and workshops
- ▶ Working with REST APIs
- ▶ Working with HTTP APIs
- ▶ Working with WebSocket APIs
- ▶ API Gateway ARNs
- ▶ OpenAPI extensions
- ▶ Security
- ▶ Tagging
- ▶ API references
- ▶ Quotas and important notes
- ▶ Document history
- ▶ AWS glossary

Step 3: Test your API

Next, you test your API to make sure that it's working. For simplicity, use a web browser to invoke your API.

To test your API

1. Sign in to the API Gateway console at <https://console.aws.amazon.com/apigateway>.
2. Choose your API.
3. Note your API's invoke URL.



The screenshot shows the AWS API Gateway console. A modal window at the top left says "Successfully created API my-http-api (abcdef123)". Below it, the "my-http-api" API details page is shown. It lists the API ID as "abcdef123", Protocol as "HTTP", and Created as "2020-12-02". The Description is "No Description" and the Default endpoint is "Enabled". At the bottom, the "my-http-api" resources page shows a single stage named "\$default" with an Invoke URL of "https://abcdef123.execute-api.us-east-2.amazonaws.com". The URL is highlighted with a yellow box.

4. Copy your API's invoke URL, and enter it in a web browser. Append the name of your Lambda function to your invoke URL to call your Lambda function. By default, the API Gateway console creates a route with the same name as your Lambda function, `my-function`.
The full URL should look like `https://abcdef123.execute-api.us-east-2.amazonaws.com/my-function`.
5. Verify your API's response. You should see the text "Hello from Lambda!" in your browser.

Lecture outcomes

- History
- Cloud Models
- Serverless Types
- Google Cloud Run
- AWS

