



# Week 10 DevOps

Dr Zhi Zhang

# Overview

- DevOps
- Fabric
- AWS Lambda

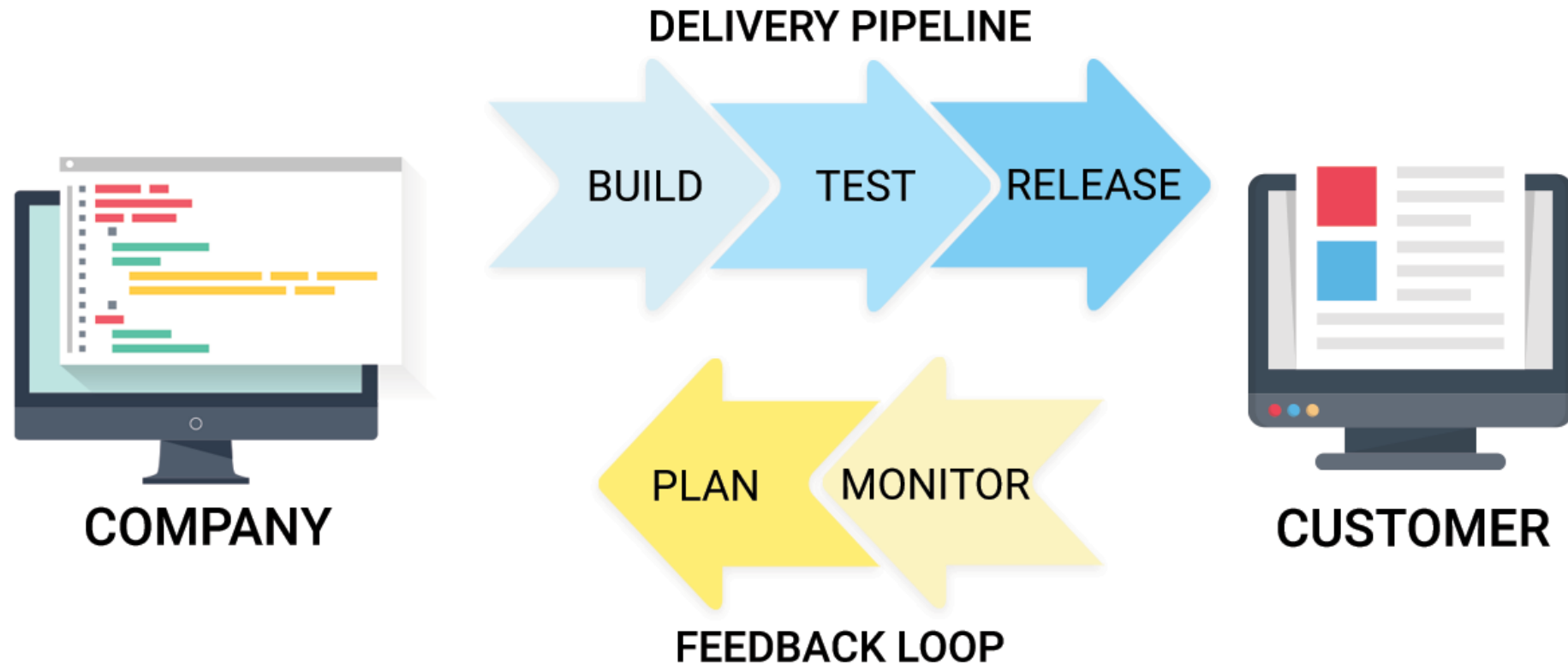
# What is DevOps?

- It is a combination of cultural philosophies, practices and tools created to facilitate organizations in delivering services/applications much faster than they can through traditional software development.

# What Is DevOps?



# HOW DEVOPS WORKS



# DevOps best practices

- Microservice
- Monitoring and Logging
- Continuous Integration
- Continuous Delivery
- Continuous Deployment

# DevOps best practices

- Microservice
  - A design approach to build a single application as a set of small services. Each service runs in its own process and communicates with other services through network.
  - Applications are broken into many individual microservices with each microservice scoped to a single purpose or function
  - **Makes the application flexible and enable frequent and small updates.**
  - **Example:** AWS Lambda.

# DevOps best practices

- Monitoring and Logging
  - Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user.
  - Tracks the performance of applications and infrastructure and detects real-time problems.
  - **Example:** AWS CloudWatch.



# DevOps best practices

- Continuous Integration
- Continuous Delivery
- Continuous Deployment

# The software/application release process:

- **Source Control:**
  - Developers use a VCS (Version Control System) to manage and track changes to a code repository.
- Run Build and Unit Tests:
- Deploy to Test Environment:
- Deploy to Production Environment:

# The software/application release process:

- Source Control:
- **Run Build and Unit Tests:**
  - Build refers to automatically compiling the code, resolving dependencies, and generating executables for the application.
  - Automated unit tests are executed to ensure that individual components of the application function correctly.
  - Failures in build and unit tests alert the developers.
- Deploy to Test Environment:
- Deploy to Production Environment:

# The software/application release process:

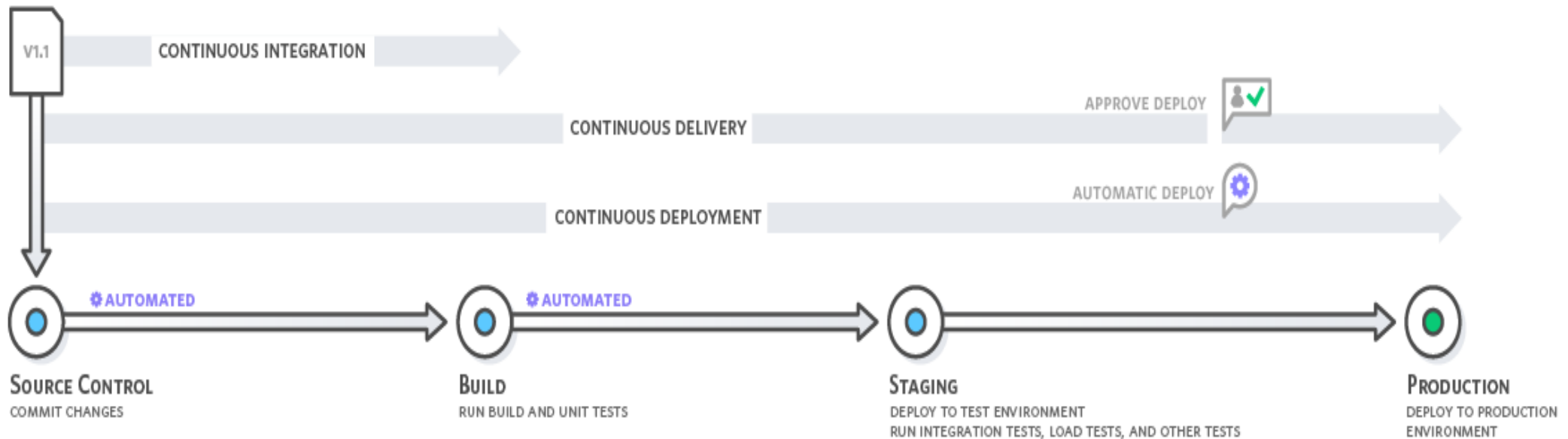
- Source Control:
- Run Build and Unit Tests:
- **Deploy to Test Environment:**
  - Automated deployment to the test environment.
  - Test Environment: a dedicated test environment mirrors the production environment but is isolated for testing purposes.
  - Test includes functional testing, integration testing, workload testing, etc.
- Deploy to Production Environment:

# The software/application release process:

- Source Control:
- Run Build and Unit Tests:
- Deploy to Test Environment:
- **Deploy to Production Environment:**
  - The application is deployed to the production environment for serving real users.

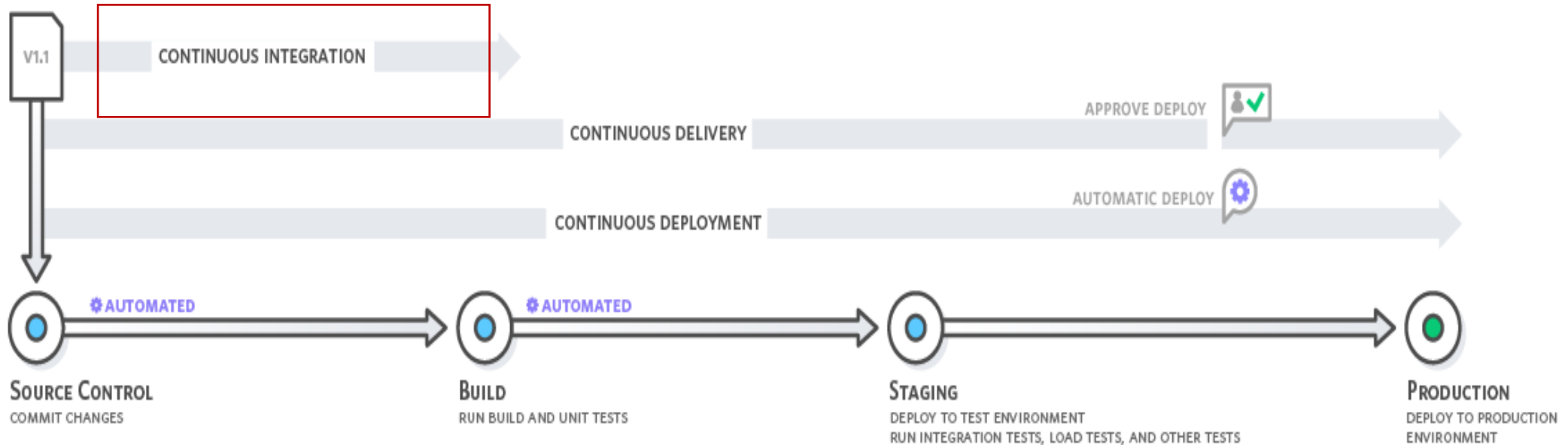
# The software/application release process:

- Source Control:
- Run Build and Unit Tests:
- Deploy to Test Environment:
- **Deploy to Production Environment:**



# DevOps best practices

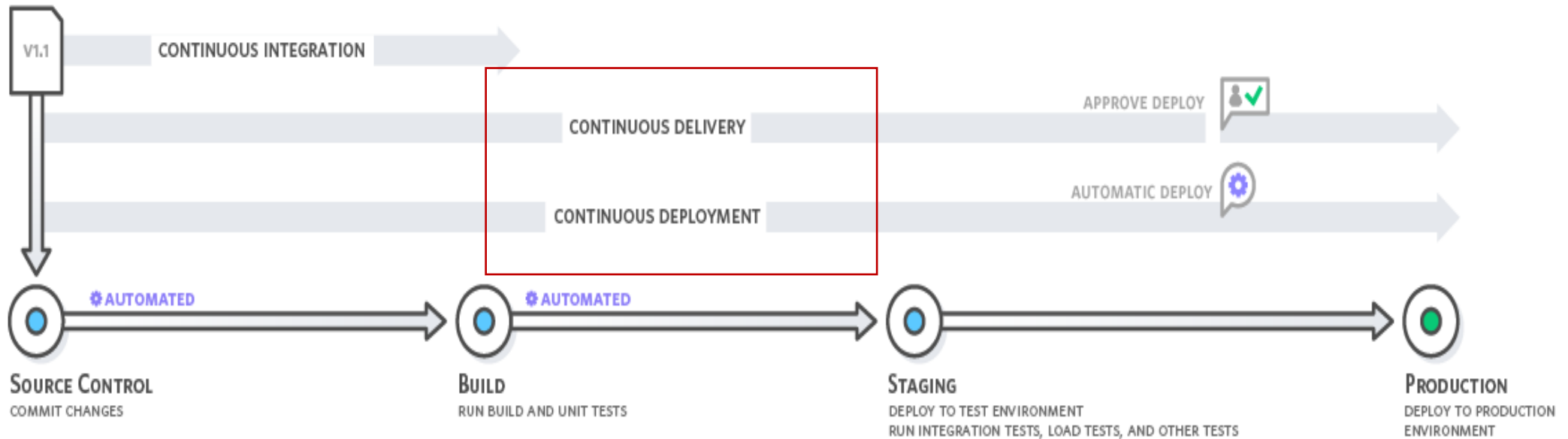
- Continuous Integration
  - Every time developers merge their code changes into a central repository, automated builds and tests start: Whenever a change is committed, the automated build and test will be triggered.



- **Benefits:** Improve Developer Productivity, Find and Address Bugs Quicker, Deliver Updates Faster

# DevOps best practices

- Continuous Delivery and Continuous Deployment:
  - Expand upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage.



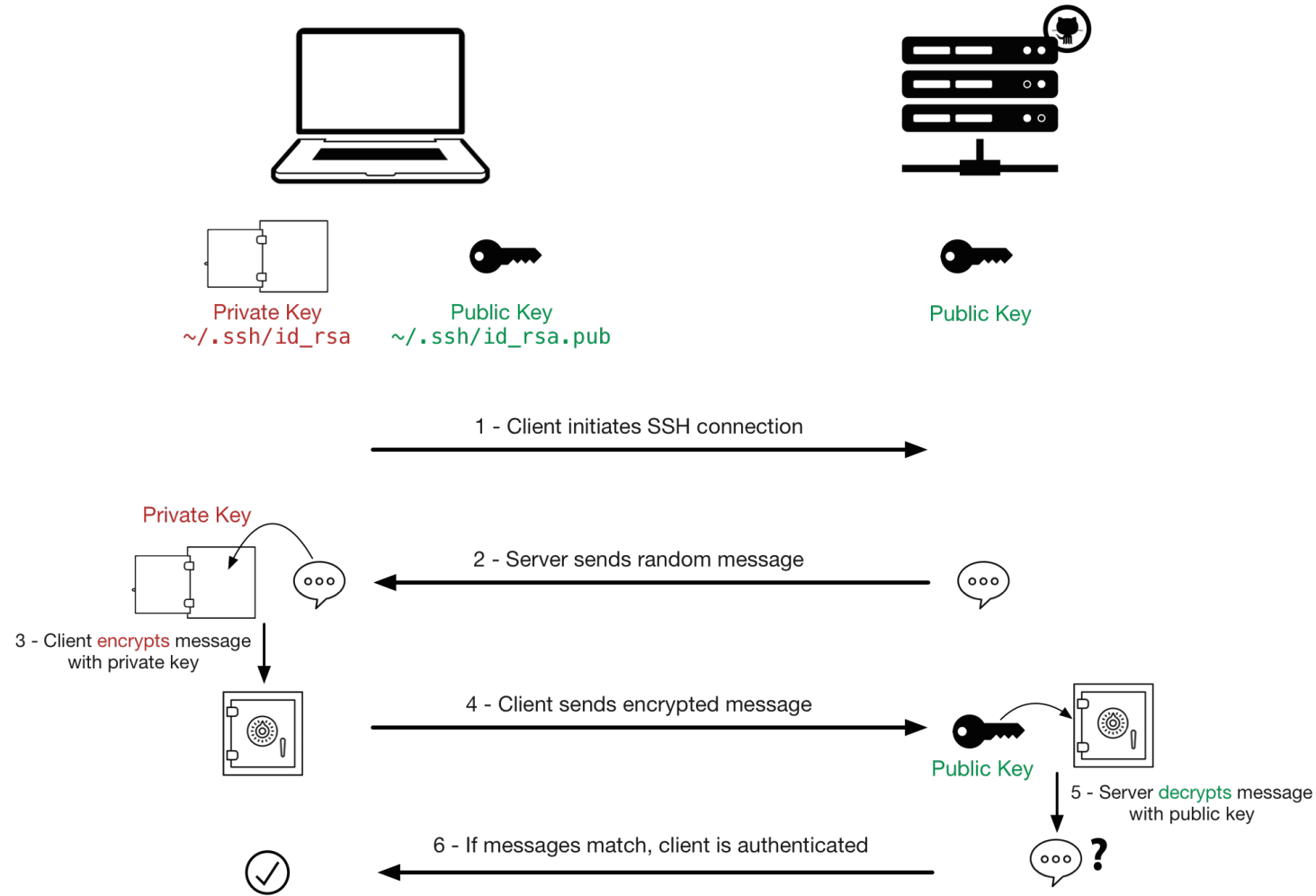
- **Benefits:** Continuous Integration + Automate the Software Release Process



# Fabric: automate tasks in DevOps

- Fabric is a high level Python library designed to execute shell commands remotely over SSH, yielding useful Python objects in return.
- OpenSSH: a widely used version of the SSH protocol, available on Mac, Linux/Unix and Windows.
- **Question**: how is OpenSSH used for user/client authentication?

# User/Client Authentication in SSH



# Add the public key to GitHub

1. Check for existing OpenSSH keys:

```
ls ~/.ssh
```

2. Generate OpenSSH keys if no key pairs exist:

```
ssh-keygen -t rsa -b 4096 -C email@example.com
```


private key: id\_rsa


public key: id\_rsa.pub


3. Add the Public Key to GitHub:


Access


 Billing and plans 

 Emails

 Password and authentication

 Sessions

 SSH and GPG keys

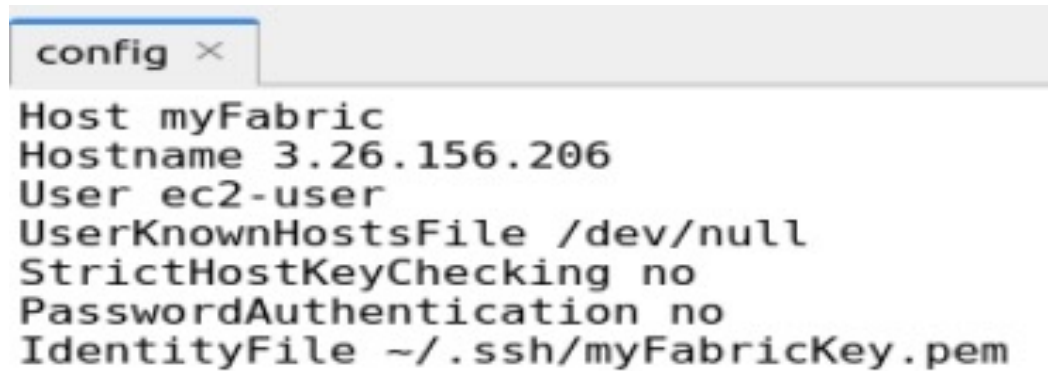
 Organizations

 Enterprises

 Moderation 

# How to configure OpenSSH to support Fabric

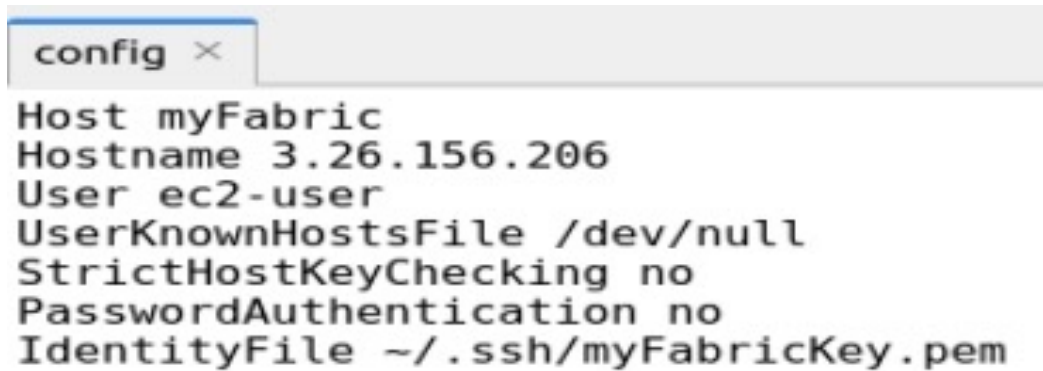
- Install fabric using: `pip install fabric`
- Create a config file in `~/.ssh`

A screenshot of a text editor window titled 'config' with a close button 'x'. The window contains the following SSH configuration text:

```
Host myFabric
Hostname 3.26.156.206
User ec2-user
UserKnownHostsFile /dev/null
StrictHostKeyChecking no
PasswordAuthentication no
IdentityFile ~/.ssh/myFabricKey.pem
```

# How to configure OpenSSH to support Fabric

- Install fabric using: `pip install fabric`
- Create a config file in `~/.ssh`



```
config ×
Host myFabric
  Hostname 3.26.156.206
  User ec2-user
  UserKnownHostsFile /dev/null
  StrictHostKeyChecking no
  PasswordAuthentication no
  IdentityFile ~/.ssh/myFabricKey.pem
```

**Question:** Some settings in the configuration above will make the SSH connection NOT secure? What are they? Justify your answer.

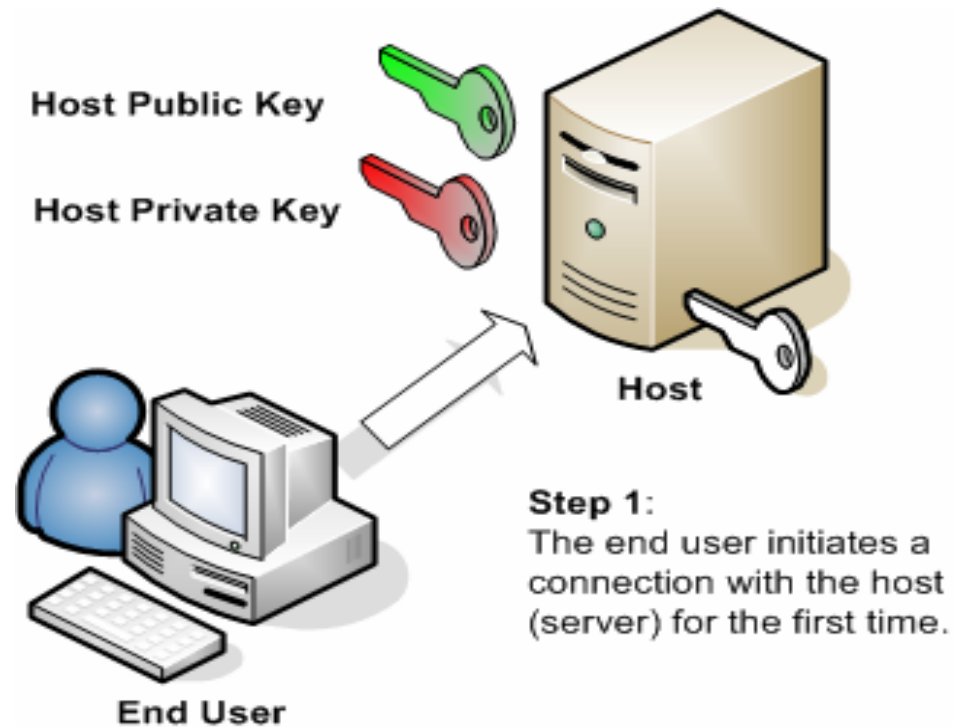
Both “UserKnownHostsFile /dev/null” and “StrictHostKeyChecking yes” have disabled host key checking, making the connection unsecure to the client.

To explain why, we need to be aware of “server/host authentication”.

# Server/Host Authentication in SSH

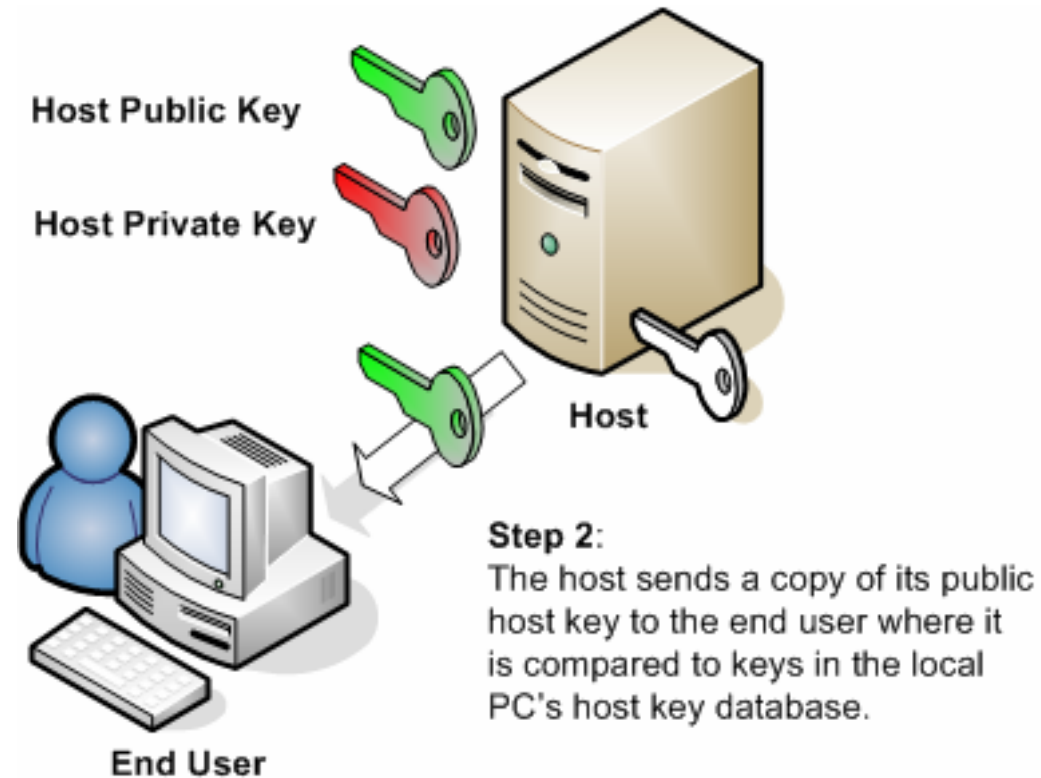
**Key pair:** host key is a key pair. Public host key is stored on and distributed to different clients, and private key is stored on the host/server.

**Key Exchange:** when a client attempts to connect to a server for the first time



# Server/Host Authentication in SSH

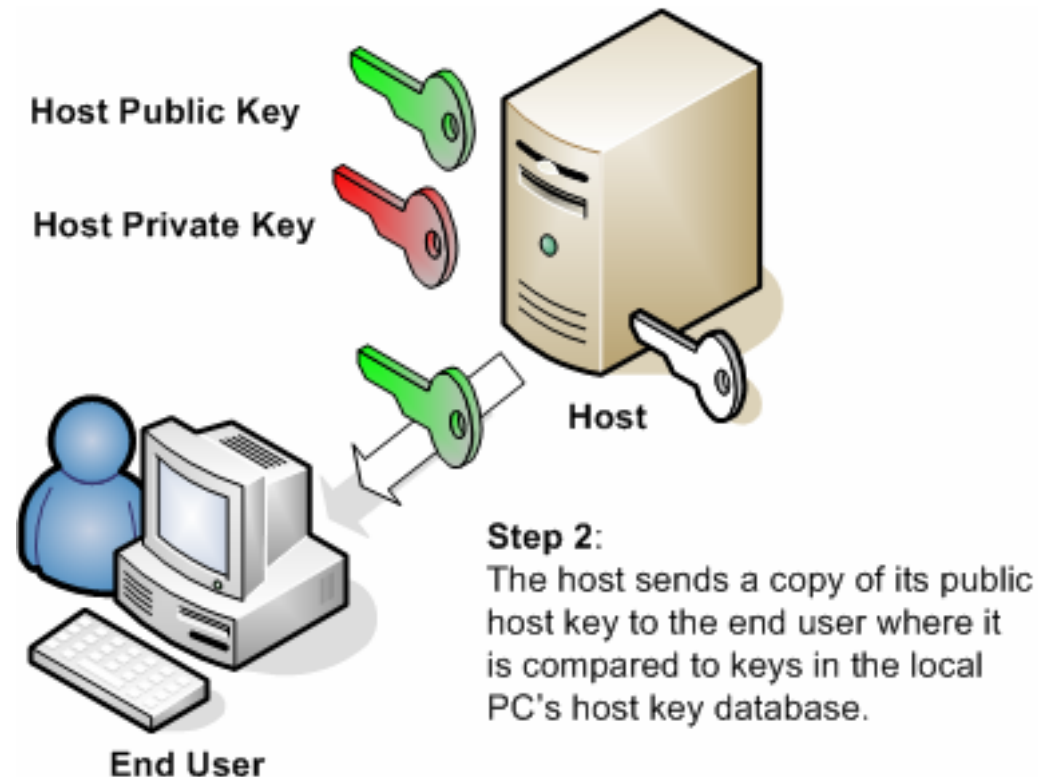
**Key Exchange:** when a client attempts to connect to a server for the first time, the server presents its host public key to the client.



# Server/Host Authentication in SSH

The host-key database is called *known\_hosts* in Linux and contains *known host keys*.

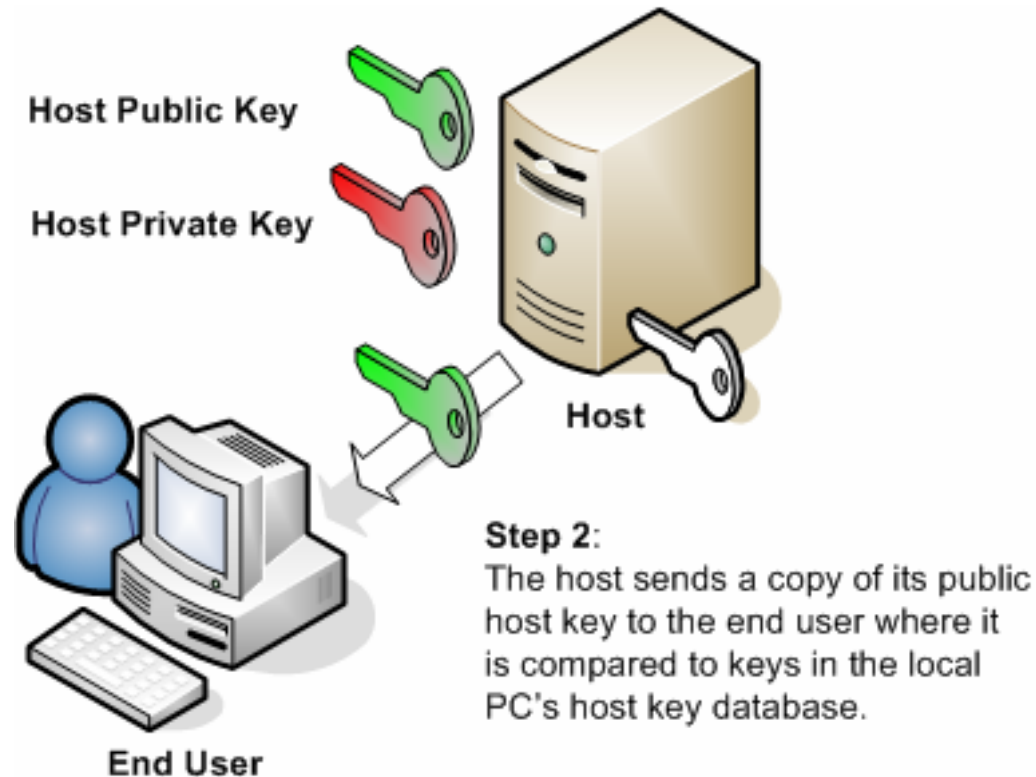
```
cits1003@cits1003-virtualbox:~$ ls -al ~/.ssh/known_hosts
-rw----- 1 cits1003 cits1003 1342 Sep 28 22:48 /home/cits1003/.ssh
/known_hosts
cits1003@cits1003-virtualbox:~$ █
```





# Server/Host Authentication in SSH

**Host Key Checking:** The client checks the server's host key against a copy stored in its "known\_hosts" file. If the check succeeds, the server is authenticated.



# Enable Server/Host Authentication in SSH

```
config ×  
Host myFabric  
Hostname 3.26.156.206  
User ec2-user  
UserKnownHostsFile /dev/null  
StrictHostKeyChecking no  
PasswordAuthentication no  
IdentityFile ~/.ssh/myFabricKey.pem
```



```
config ×  
Host myFabric  
Hostname 3.26.156.206  
User ec2-user  
StrictHostKeyChecking yes  
PasswordAuthentication no  
IdentityFile ~/.ssh/myFabricKey.pem
```

# Fabric: common functions

```
from fabric import Connection  
c = Connection('myFabric')
```

- upload a local file to the remote server
  - `c.put(localfile, remotefilepath)`
- run a remote command
  - `c.run(command, otherargs)`
- run a remote command with sudo
  - `c.sudo(command, otherargs)`

# Fabric: common functions

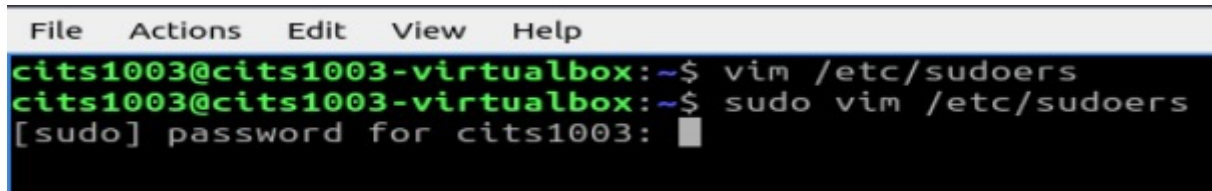
```
from fabric import Connection  
c = Connection('myFabric')
```

- upload a local file to the remote server
  - `c.put(localfile, remotefilepath)`
- run a remote command
  - `c.run(command, otherargs)`
- run a remote command with sudo
  - `c.sudo(command, otherargs)`

**Question:** what are the differences between 'sudo' and 'su'?

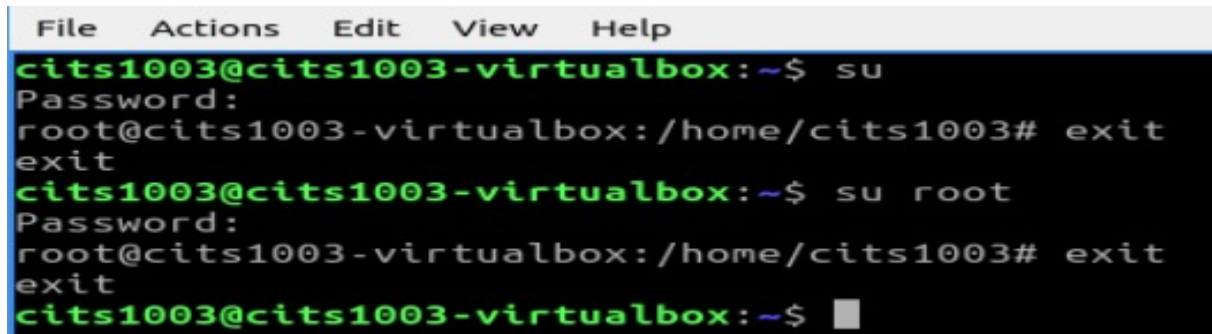
**Question:** what are the differences between 'sudo' and 'su' ?

- sudo (Superuser Do): allows a user to execute specific commands that require superuser privileges.

A terminal window with a menu bar (File, Actions, Edit, View, Help) and a dark background. The prompt is 'cits1003@cits1003-virtualbox:~\$'. The user enters 'vim /etc/sudoers'. The prompt changes to 'cits1003@cits1003-virtualbox:~\$' and the user enters 'sudo vim /etc/sudoers'. The prompt changes to '[sudo] password for cits1003:' followed by a cursor.

```
File  Actions  Edit  View  Help
cits1003@cits1003-virtualbox:~$ vim /etc/sudoers
cits1003@cits1003-virtualbox:~$ sudo vim /etc/sudoers
[sudo] password for cits1003: █
```

- su (Switch User): allows a user to switch to another user account by entering that account's password.

A terminal window with a menu bar (File, Actions, Edit, View, Help) and a dark background. The prompt is 'cits1003@cits1003-virtualbox:~\$'. The user enters 'su'. The prompt changes to 'root@cits1003-virtualbox:/home/cits1003#'. The user enters 'exit'. The prompt returns to 'cits1003@cits1003-virtualbox:~\$'. The user enters 'su root'. The prompt changes to 'root@cits1003-virtualbox:/home/cits1003#'. The user enters 'exit'. The prompt returns to 'cits1003@cits1003-virtualbox:~\$' followed by a cursor.

```
File  Actions  Edit  View  Help
cits1003@cits1003-virtualbox:~$ su
Password:
root@cits1003-virtualbox:/home/cits1003# exit
cits1003@cits1003-virtualbox:~$ su root
Password:
root@cits1003-virtualbox:/home/cits1003# exit
cits1003@cits1003-virtualbox:~$ █
```

fabfile.py: a Python script that is a collection of tasks and functions

```
fabfile.py ×
1 from fabric import task
2
3
4 @task
5 def fileOps(c):
6     if c.run('test -f ~/myFabricFile', warn=True).failed:
7         c.put('myFabricFile.tgz', '/home/ec2-user')
8         c.run('tar -C ~/ -xf /home/ec2-user/myFabricFile.tgz')
9
10 @task
11 def sudoOps(c):
12     c.sudo('cat /etc/passwd')
13
```

@task is a decorator from the Fabric library.

**Question:** what does the code snippet above primarily do?

```
fabfile.py x
1 from fabric import task
2
3
4 @task
5 def fileOps(c):
6     if c.run('test -f ~/myFabricFile', warn=True).failed:
7         c.put('myFabricFile.tgz', '/home/ec2-user')
8         c.run('tar -C ~/ -xf /home/ec2-user/myFabricFile.tgz')
9
10 @task
11 def sudoOps(c):
12     c.sudo('cat /etc/passwd')
13
```

@task is a decorator from the Fabric library.

**Question:** what does the code snippet above primarily do?

A task named fileOps is defined. It first checks if a file named myFabricFile exists in the home directory of the remote host. If the check does not exist, the warn=True argument prevents the check from aborting, and instead will return a warning. Besides, the task uploads a file named myFabricFile.tgz to the remote host's directory of '/home/ec2-user'. Then, it extracts the contents of myFabricFile.tgz into the remote host's home directory.

The other task named sudoOps is defined: This task executes the 'cat /etc/passwd command' with elevated privileges.

# Execute fabfile.py

```
fabfile.py x
1 from fabric import task
2
3
4 @task
5 def fileOps(c):
6     if c.run('test -f ~/myFabricFile', warn=True).failed:
7         c.put('myFabricFile.tgz', '/home/ec2-user')
8         c.run('tar -C ~/ -xf /home/ec2-user/myFabricFile.tgz')
9
10 @task
11 def sudoOps(c):
12     c.sudo('cat /etc/passwd')
13
```

Navigate to the directory where the fabfile.py resides:

```
cits1003@cits1003-virtualbox:~$ ls -al fabfile.py
-rw-rw-r-- 1 cits1003 cits1003 231 Sep 28 23:23 fabfile.py
cits1003@cits1003-virtualbox:~$ fab -l
Available tasks:

fileOps
sudoOps
```

```
cits1003@cits1003-virtualbox:~$ fab -H ec2-user@52.62.6.162 -i ~/.ssh/myFabricKey.pem sudoOps
```



# AWS Lambda: automate tasks in DevOps

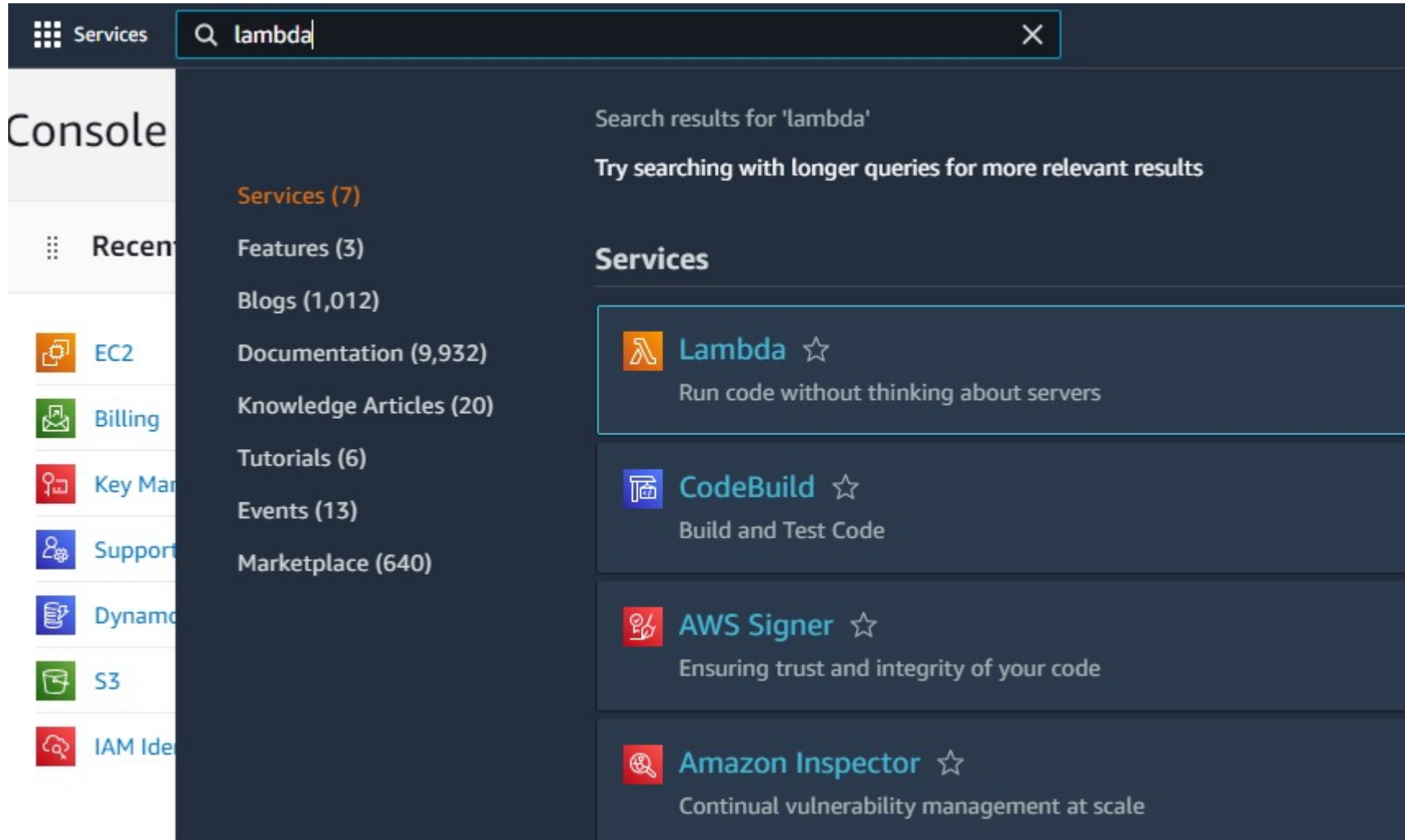
- Lambda is a compute service that lets us run code without provisioning or managing servers. With Lambda, all we need to do is **supply our code in one of the language runtimes that Lambda supports**.
- Lambda runs our code automatically on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and logging.

# AWS Lambda: automate tasks in DevOps

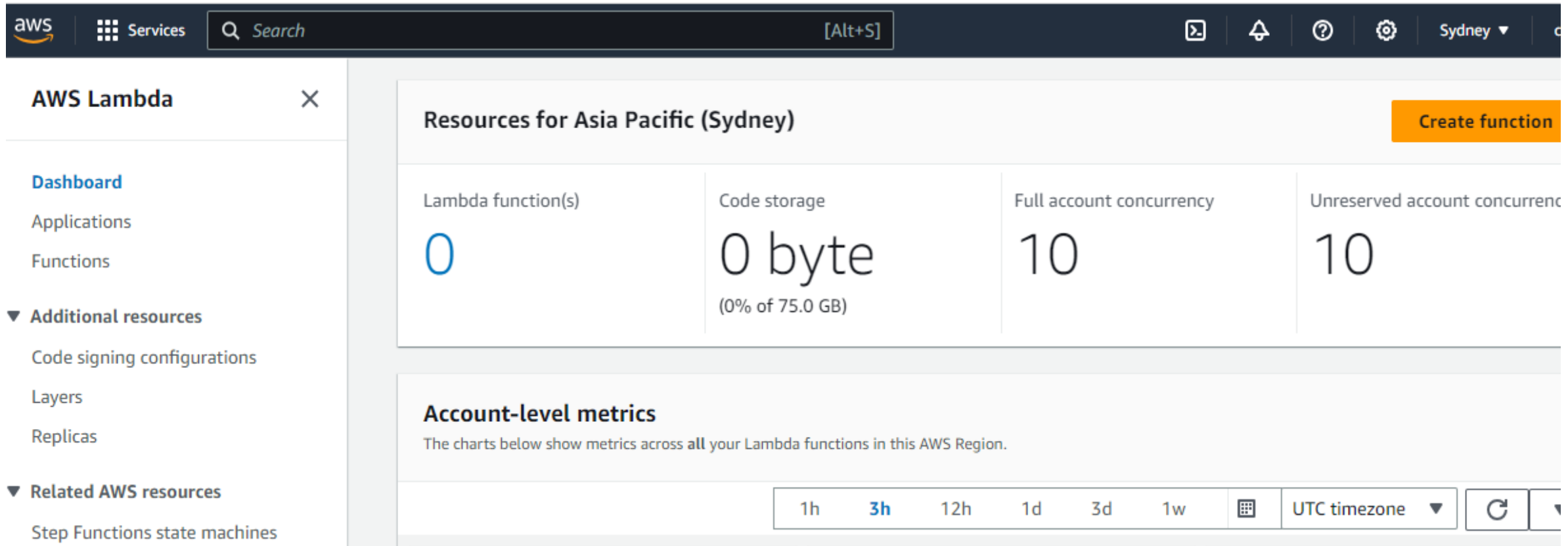
## Examples:

- A toy lambda function
- A combination of lambda function and S3

# Step1: open the AWS Lambda Console



# Step1: open the AWS Lambda Console



The screenshot displays the AWS Lambda console interface. The top navigation bar includes the AWS logo, a 'Services' menu, a search bar, and a keyboard shortcut '[Alt+S]'. On the right side of the header, there are icons for a dashboard, notifications, help, settings, and the current region 'Sydney'.

The left-hand sidebar is titled 'AWS Lambda' and contains a list of navigation options: 'Dashboard' (highlighted in blue), 'Applications', 'Functions', 'Additional resources' (expanded), and 'Related AWS resources'. Under 'Additional resources', there are links for 'Code signing configurations', 'Layers', and 'Replicas'. Under 'Related AWS resources', there is a link for 'Step Functions state machines'.

The main content area is titled 'Resources for Asia Pacific (Sydney)' and features a prominent orange 'Create function' button in the top right corner. Below the title, there is a table showing resource usage:

Lambda function(s)	Code storage	Full account concurrency	Unreserved account concurrency
0	0 byte (0% of 75.0 GB)	10	10

Below the resource table, there is a section titled 'Account-level metrics' with a subtitle: 'The charts below show metrics across all your Lambda functions in this AWS Region.' At the bottom of this section, there is a row of controls for the metrics charts, including time range selectors (1h, 3h, 12h, 1d, 3d, 1w), a calendar icon, a 'UTC timezone' dropdown menu, a refresh icon, and a close icon.

## Step2: select a Lambda blueprint and configure it

[Lambda](#) > [Functions](#) > Create function

### Create function [Info](#)

AWS Serverless Application Repository applications have moved to [Create application](#).

☐ Author from scratch  
Start with a simple Hello World example.

☒ Use a blueprint  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ Container image  
Select a container image to deploy for your function.

# Step3: basic information configuration

**Basic information** [Info](#)

Blueprint name

Hello world function  
A starter AWS Lambda function.

python3.7 ▼

Function name

Enter a name that describes the purpose of your function.

Hello\_CITS5503

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime

python3.7

Architecture

x86\_64

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions


☐ Use an existing role

☒ Create a new role from AWS policy templates

- Execution role: an IAM role that grants a Lambda function permission to access AWS services and resources

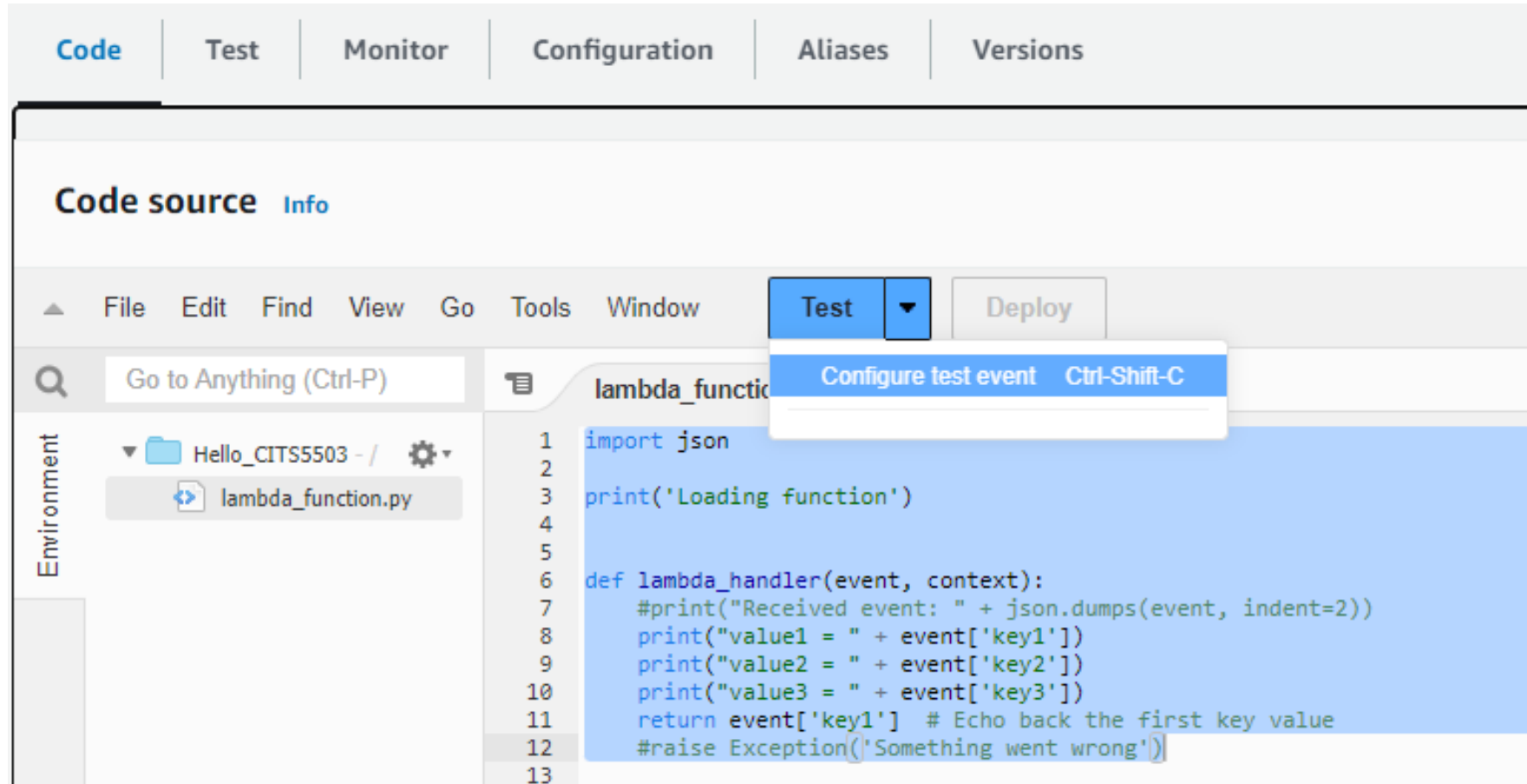
# Step3: basic information configuration

## Lambda function code

Code is preconfigured by the chosen blueprint. You can configure it after you create the function. [Learn more](#)  about deploying Lambda functions.

```
1 import json
2
3 print('Loading function')
4
5
6 def lambda_handler(event, context):
7     #print("Received event: " + json.dumps(event, indent=2))
8     print("value1 = " + event['key1'])
9     print("value2 = " + event['key2'])
10    print("value3 = " + event['key3'])
11    return event['key1'] # Echo back the first key value
12    #raise Exception('Something went wrong')
13
```

## Step4: invoke Lambda function and verify results





# Step4: invoke Lambda function and verify results

## Configure test event ✕

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

☒ Create new event

☐ Edit saved event

Event name

HelloCITS5503Event

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

☒ Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#) 

☐ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#) 

# Step4: invoke Lambda function and verify results

Template - optional

hello-world

Event JSON

Format JSON

```
1 {  
2   "key1": "value1",  
3   "key2": "value2",  
4   "key3": "value3"  
5 }
```

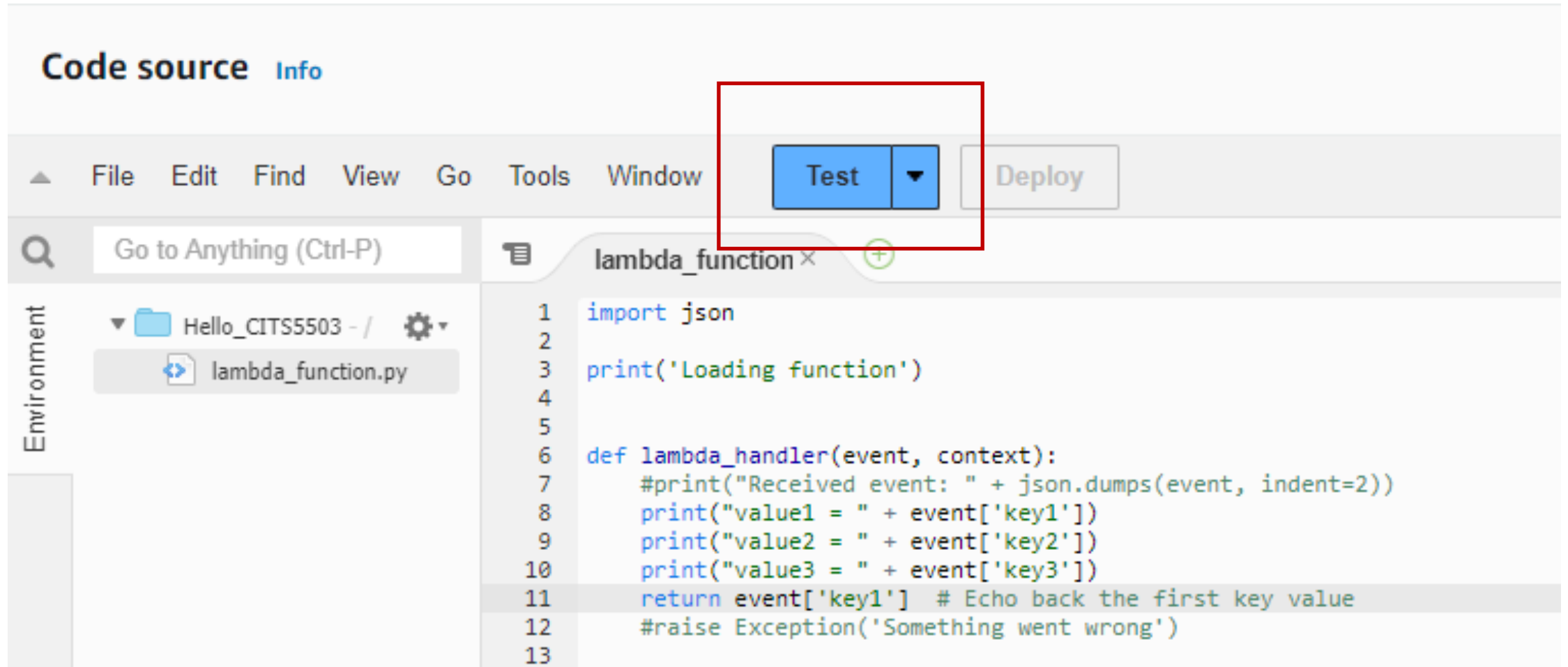


Event JSON

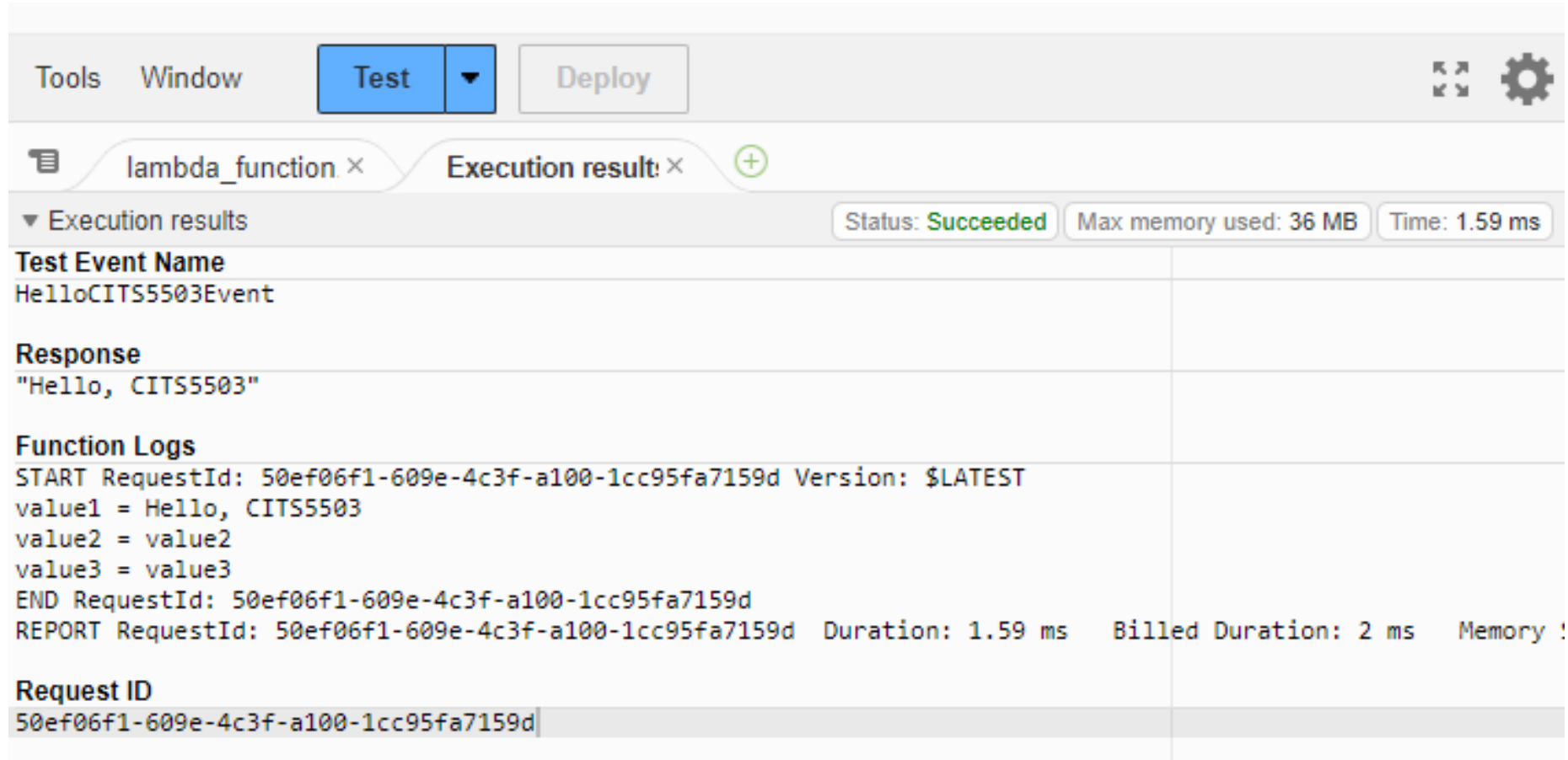
Format JSON

```
1 {  
2   "key1": "Hello, CITS5503",  
3   "key2": "value2",  
4   "key3": "value3"  
5 }
```

## Step4: invoke Lambda function and verify results



## Step4: invoke Lambda function and verify results



The screenshot displays the AWS Lambda console interface. At the top, there are tabs for 'Tools', 'Window', 'Test', and 'Deploy'. The 'Test' tab is active. Below the tabs, there are two tabs: 'lambda\_function' and 'Execution result:'. The 'Execution result:' tab is selected, showing a green plus icon. The execution results are displayed in a table-like format with the following sections:

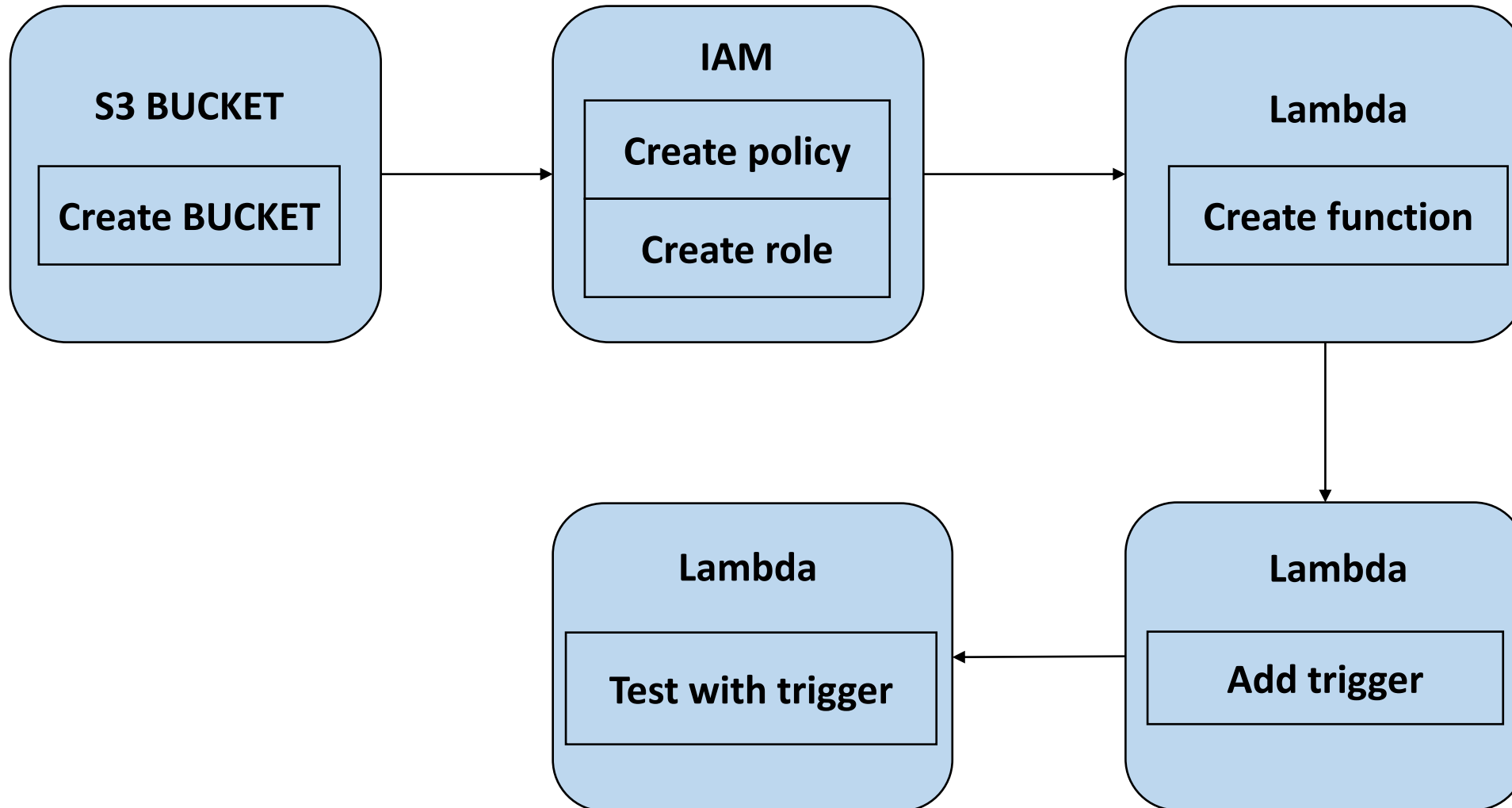
Execution results	
Status: <b>Succeeded</b> Max memory used: 36 MB Time: 1.59 ms	
<b>Test Event Name</b>	
HelloCITS5503Event	
<b>Response</b>	
"Hello, CITS5503"	
<b>Function Logs</b>	
START RequestId: 50ef06f1-609e-4c3f-a100-1cc95fa7159d Version: \$LATEST value1 = Hello, CITS5503 value2 = value2 value3 = value3 END RequestId: 50ef06f1-609e-4c3f-a100-1cc95fa7159d REPORT RequestId: 50ef06f1-609e-4c3f-a100-1cc95fa7159d Duration: 1.59 ms Billed Duration: 2 ms Memory :	
<b>Request ID</b>	
50ef06f1-609e-4c3f-a100-1cc95fa7159d	

# AWS Lambda: automate tasks in DevOps

## Example:

- A toy Lambda function
- A combination of lambda and S3

# Combine Lambda with S3



# Step1: create an Amazon S3 bucket and upload the test object





[Amazon S3](#) > [Buckets](#) > cits5503s3lambda


## cits5503s3lambda [Info](#)


[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#)

**Objects (1)**

Objects are the fundamental entities stored in Amazon S3. You can use permissions. [Learn more](#) [↗](#)

  [Copy S3 URI](#)  [Copy URL](#) 



<input type="checkbox"/>	Name	Type
<input type="checkbox"/>	 <a href="#">UWA_campus.jpg</a>	jpg

## Step2: create a permissions policy

Policy name	Type	Used as
<input type="radio"/>  <a href="#">s3_trigger_lambda_cits5503</a>	Customer managed	Permissions policy (1)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::*/*"
    }
  ]
}
```



## Step2: create a permissions policy

Policy name	Type	Used as
 s3_trigger_lambda_cits5503	Customer managed	Permissions policy (1)

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3::*/*"
    }
  ]
}
```

A log stream is a sequence of log events that share the same source. Each separate source of logs in cloudwatch logs makes up a separate log stream.

A log group is a group of log streams that share the same monitoring settings.

This permissions policy allows read actions against s3 buckets and write actions against cloudwatch logs.

# Step3: create an execution role

[IAM](#) > [Roles](#) > Create role

Step 1

Select trusted entity

Step 2

Add permissions

Step 3

Name, review, and create

## Select trusted entity [Info](#)

### Trusted entity type

☒ **AWS service**  
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

☐ **AWS account**  
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

☐ **Web identity**  
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

☐ **SAML 2.0 federation**  
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

☐ **Custom trust policy**  
Create a custom trust policy to enable others to perform actions in this account.

## Step3: create an execution role

### Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case

Lambda ▼

Choose a use case for the specified service.

Use case

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

## Add permissions [Info](#)

### Permissions policies (1/893) [Info](#)

Choose one or more policies to attach to your new role.

X

Filter by Type

All types ▼

1 match

<input checked="" type="checkbox"/>	Policy name <a href="#">↗</a>	Type ▲ ▼	Description
<input checked="" type="checkbox"/>	<a href="#">s3_trigger_lambda_cits5503</a> +	Customer managed	-

## Step3: create an execution role

### Role details

#### Role name

Enter a meaningful name to identify this role.

role\_s3\_trigger\_lambda\_cits5503

Maximum 64 characters. Use alphanumeric and '+=, @-\_' characters.

#### Description

Add a short explanation for this role.

Allows Lambda functions to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+=, @-\_' characters.

# Step4: create the lambda function

[Lambda](#) > [Functions](#) > Create function

## Create function [Info](#)

AWS Serverless Application Repository applications have moved to [Create application](#).

☒ Author from scratch  
Start with a simple Hello World example.

☐ Use a blueprint  
Build a Lambda application from sample code and configuration presets for common use cases.

### Basic information

Function name  
Enter a name that describes the purpose of your function.

func\_s3\_trigger\_lambda\_cits5503

Use only letters, numbers, hyphens, or underscores with no spaces.

# Step4: create the lambda function

## Runtime [Info](#)

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.10



## Architecture [Info](#)

Choose the instruction set architecture you want for your function code.

☒ x86\_64

☐ arm64

## Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

### ▼ Change default execution role

#### Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions

☒ Use an existing role

☐ Create a new role from AWS policy templates

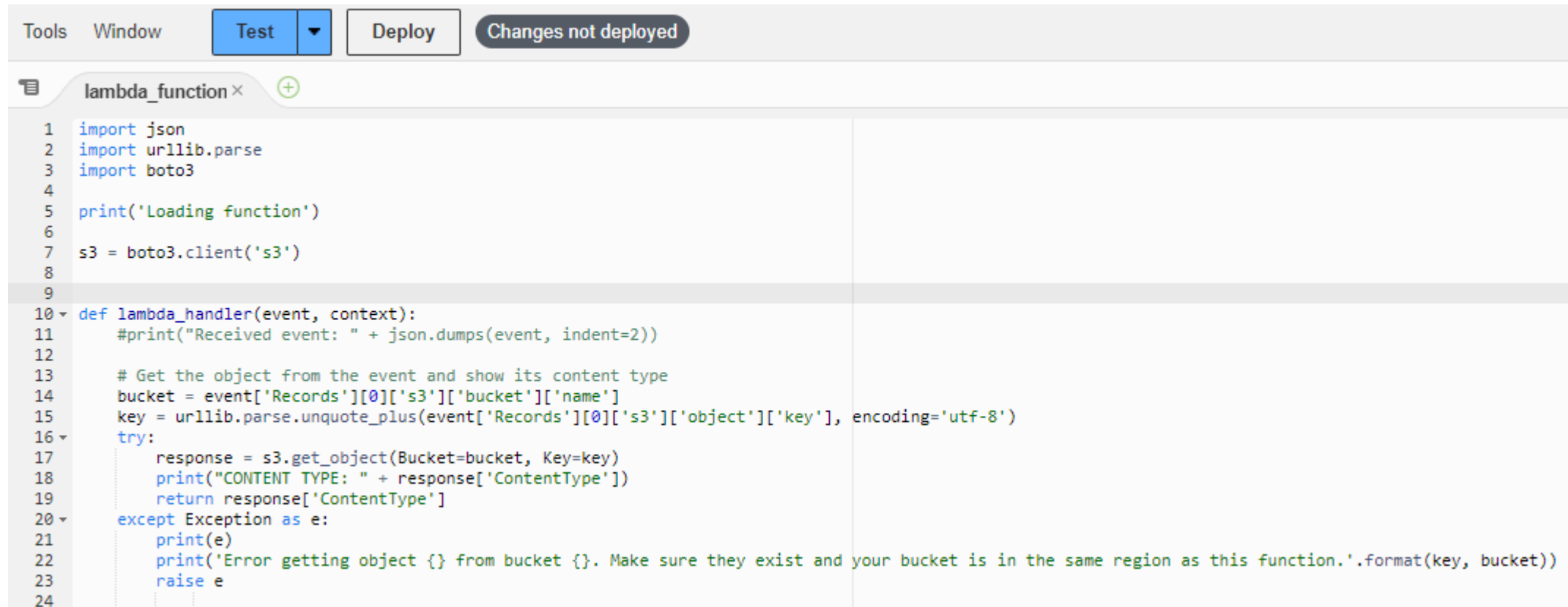
#### Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

role\_s3\_trigger\_lambda\_cits5503



## Step4: create the lambda function



The screenshot shows a code editor interface with a top bar containing 'Tools', 'Window', 'Test', 'Deploy', and a 'Changes not deployed' status. The editor has a tab labeled 'lambda\_function' with a close button and a plus icon. The code is written in Python and defines a lambda handler that interacts with an S3 bucket. The code is as follows:

```
1 import json
2 import urllib.parse
3 import boto3
4
5 print('Loading function')
6
7 s3 = boto3.client('s3')
8
9
10 def lambda_handler(event, context):
11     #print("Received event: " + json.dumps(event, indent=2))
12
13     # Get the object from the event and show its content type
14     bucket = event['Records'][0]['s3']['bucket']['name']
15     key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
16     try:
17         response = s3.get_object(Bucket=bucket, Key=key)
18         print("CONTENT TYPE: " + response['ContentType'])
19         return response['ContentType']
20     except Exception as e:
21         print(e)
22         print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same region as this function.'.format(key, bucket))
23         raise e
24
```

## Step4: create the lambda function

```
13  
14 bucket = event['Records'][0]['s3']['bucket']['name']  
15
```

1. event['Records']: contains an array of records. Each record corresponds to a specific event that triggered the lambda function.
2. event['Records'][0]: retrieves the first record in this array.
3. event['Records'][0]['s3']: contains information specific to an S3 event in the record.
4. event['Records'][0]['s3']['bucket']: contains information about the S3 bucket where the S3 event occurred.
5. event['Records'][0]['s3']['bucket']['name']: retrieves the name field from the bucket.



## Step4: create the lambda function

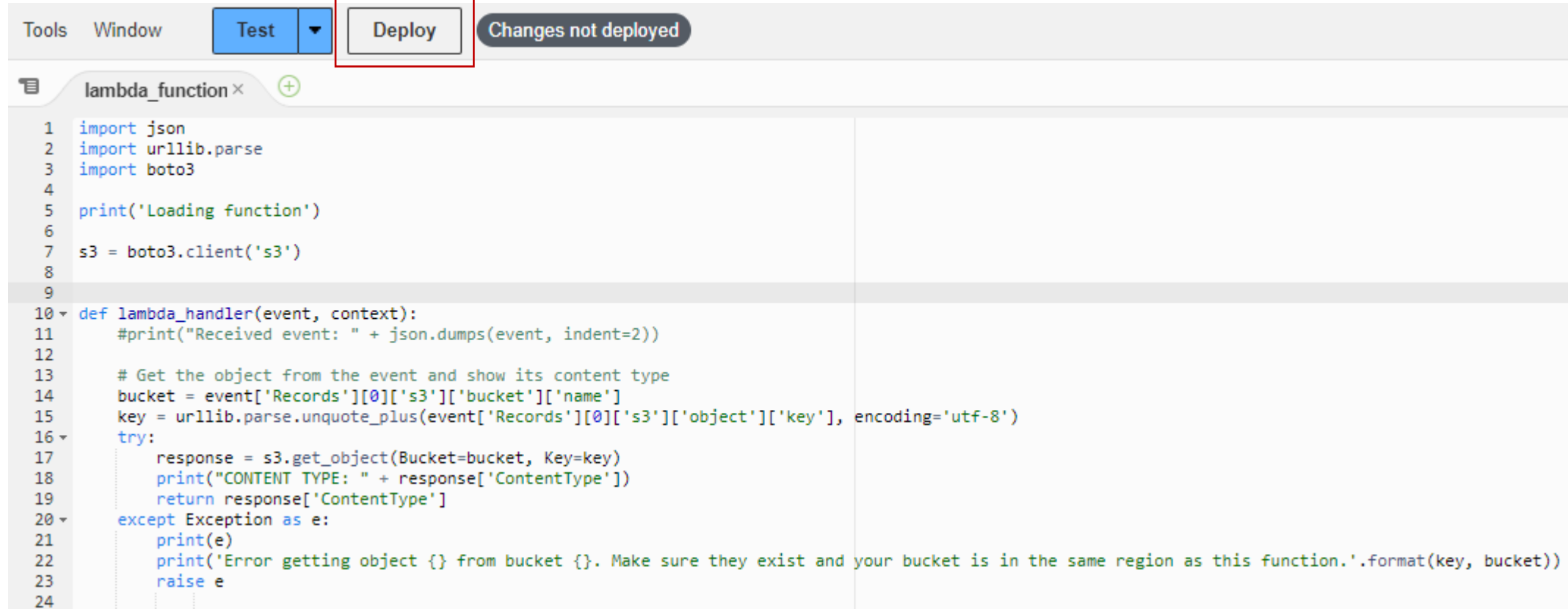
```
14  
15     key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')  
16
```

1. `event['Records'][0]['s3']['object']['key']`: retrieves the S3 object key from the S3 object, encoded by utf-8.
2. `urllib.parse.unquote_plus()`: a function call that decodes an encoded URL string.
3. `encoding='utf-8'`: specifies the character encoding to be used when decoding the object key.

```
15  
16     print("key url: " + event['Records'][0]['s3']['object']['key'])  
17     key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')  
18     print("decoded key url: " + key)
```

▶	2023-10-02T14:53:09.908+08:00	key url: test.txt
▶	2023-10-02T14:53:09.908+08:00	decoded key url: test.txt

## Step4: create the lambda function



The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'Tools' and 'Window'. Below these, there are buttons for 'Test' and 'Deploy'. The 'Deploy' button is highlighted with a red rectangle. To the right of the 'Deploy' button, there is a status indicator that says 'Changes not deployed'. Below the buttons, there is a tab labeled 'lambda\_function' with a close button and a plus sign. The main area displays the Python code for the lambda function. The code includes imports for 'json', 'urllib.parse', and 'boto3'. It prints 'Loading function' and creates an S3 client. The 'lambda\_handler' function is defined, which prints the received event, extracts the bucket and key from the event, and uses the S3 client to get the object's content type. It includes a try-except block to handle exceptions and print an error message if getting the object fails.

```
1 import json
2 import urllib.parse
3 import boto3
4
5 print('Loading function')
6
7 s3 = boto3.client('s3')
8
9
10 def lambda_handler(event, context):
11     #print("Received event: " + json.dumps(event, indent=2))
12
13     # Get the object from the event and show its content type
14     bucket = event['Records'][0]['s3']['bucket']['name']
15     key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
16     try:
17         response = s3.get_object(Bucket=bucket, Key=key)
18         print("CONTENT TYPE: " + response['ContentType'])
19         return response['ContentType']
20     except Exception as e:
21         print(e)
22         print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same region as this function.'.format(key, bucket))
23         raise e
24
```

## Step5: create the S3 trigger

[Lambda](#) > [Functions](#) > func\_s3\_trigger\_lambda\_cits5503

# func\_s3\_trigger\_lambda\_cits5503

▼ Function overview [Info](#)

 func\_s3\_trigger\_lambda\_cits5503


 Layers (0)

[+ Add trigger](#)

# Step5: create the S3 trigger

## Add trigger

### Trigger configuration [Info](#)

 **S3**  
aws asynchronous storage

#### Bucket

Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.

✕ ↻

Bucket region: ap-southeast-2

#### Event types

Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. For each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match a key.

All object create events ✕

# Step6: test the Lambda function with the S3 trigger

## Objects (2)

Objects are the fundamental entities stored in Amazon S3. You permissions. [Learn more](#)



Copy S3 URI

Copy URL

Find objects by prefix



Name



Type



test.txt

txt

CloudWatch > Log groups

## Log groups (3)

By default, we only load up to 10000 log groups.



Actions

View in Logs Insights

Start tailing

func



1 match



Exact match



Log group



Data prot...



Sensitive ...



Retention



/aws/lambda/func\_s3\_trigger\_lambda\_cits5503

-

-

Never expire

# Step6: test the Lambda function with the S3 trigger

Log streams

Tags

Metric filters

Subscription filters

Log streams (1)

Q

Filter log streams or try prefix search

☐

Log stream

☐ 2023/10/02/[\$LATEST]ddd91b5f8aff4e70bdea5942d6133198

▶	Timestamp	Message
		No older events at this moment. <a href="#">Retry</a>
▶	2023-10-02T13:55:21.603+08:00	INIT_START Runtime Version: python:3.10.v13 Runtime Version ARN: arn:aws:lambda:ap-southeast-2::run...
▶	2023-10-02T13:55:21.821+08:00	Loading function
▶	2023-10-02T13:55:21.953+08:00	START RequestId: cff20359-347f-4f7e-9622-610032cb62b6 Version: \$LATEST
▶	2023-10-02T13:55:22.202+08:00	CONTENT TYPE: text/plain