

Infrastructure as Code

Infrastructure as Code

Doing the same thing over and over again

- So far what we've done in AWS has been done “by hand”
- This is fine for development and experimentation
- Once you have things figured out however, you want to codify your infrastructure
 - AWS CLI
 - CloudFormation
 - Python SDK (boto3)
 - TerraForm

Infrastructure as Code

aws-cli

- On your EC2 instance, the AWS CLI is pre-installed
- You can install it on your laptop too
 - <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

Infrastructure as Code

aws-cli

- You need IAM credentials from your AWS account to use the CLI
- Log in to AWS Academy
 - <https://awsacademy.instructure.com/login/canvas>
- Start your AWS environment

Infrastructure as Code

aws-cli

- Under AWS Details
- Click on the “Show” button for AWS CLI

The screenshot shows a web browser window titled "Learner Lab" with the URL <https://awsacademy.instructure.com/courses/27873/modules/items/2315482>. The page displays a navigation sidebar on the left with links: Home, Modules, and Discussions. The main content area shows a module titled "ddd_v1_w_VDS". In the "Cloud Access" section, there is a "AWS CLI" button with a blue border, which is highlighted with a larger blue box. Below this, there is information about the session: "Remaining session time: 03:58:23(239 minutes)", "Session started at: 2022-10-20T20:47:32-0700", and "Session to end at: 2022-10-21T00:47:32-0700". It also states "Accumulated lab time: 00:11:00 (11 minutes)" and "No running instance". At the bottom, there are buttons for "SSH key Show", "Download PEM", "Download PPK", and "AWS SSO Download URL". A table at the very bottom provides account details: "AWSAccountId" (031929808245) and "Region" (us-east-1). The "AWS Details" link in the top right corner is also highlighted with a blue box.

Infrastructure as Code

aws-cli

- Copy the contents of the expanded box in to a new file in your user's home directory, inside the hidden `~/.aws/` folder named `credentials`.
- See lecture slides 07-aws for walkthrough of setting up credentials in VS Code

The screenshot shows a web browser window titled "Learner Lab" with the URL <https://awsacademy.instructure.com/courses/27873/modules/items/2315482>. The page displays a navigation sidebar on the left with links for "aws", "Account", "Dashboard", "Courses", "Calendar", "Inbox", and "History". The main content area shows a breadcrumb path: ALLv1-27873 > Modules > Learner Lab > Learner Lab. A status bar at the top indicates "AWS" with a green dot, a duration of "03:59", and buttons for "Start Lab", "End Lab", "AWS Details" (which is highlighted with a blue box), "Readme", and "R". Below the status bar, there is a section titled "Cloud Access" with the heading "AWS CLI:" and the instruction "Copy and paste the following into `~/.aws/credentials`". A code block shows the [default] section of a credentials file with several long, randomly generated AWS access keys and session tokens.

```
[default]
aws_access_key_id=ASIAQ03ZKGV2SXVEWRW4
aws_secret_access_key=sj4K8Ufaf6Fs4uRV3tTYXngSrLISwvTyTBKg
aws_session_token=FwoGZXIvYXdzEKX//////////wEaD0mtJ2WmDVqY
GSK9Ad8ssTQxMpSwJv0vCY06tVFnNcG/UX/GdLRG+lFgrbOP3GR67Z69US
8/jdqgmnZAeMN/4jtqDN8fKi40/dhNuDfivsjS2zN8Ewm8ggZ/swUlUuV7
PcPQYBCYWuVqJIXW1edQDGd21bw/p0/owzcZ+vCyW4SZL84DRi2Slnmx76
NPLyqdhL7MJ5SJmXUIUL70DdFvX9s08k52U6Yr2HN/MQ4qUH5upT986nX5
sJtPBylV2SSjVrMiaBjIt+/+TwXY00tKbPS0D28FIbSuJoFtB0IoTgNLal
```

<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html>

Session started at: 2022-10-20T20:47:32-0700

Session to end at: 2022-10-21T00:47:32-0700

Accumulated lab time: 00:11:00 (11 minutes)

No running instance

Infrastructure as Code

aws-cli

The screenshot shows the AWS Toolkit for VS Code interface. The left sidebar has icons for Explorer, Open Editors, AWS, Config, and Credentials. The 'credentials' item under 'AWS' is selected. The main area displays the contents of the '.aws/credentials' file:

```
1 [default]
2 aws_access_key_id=ASIAQ03ZKGV2SXVEWRW4
3 aws_secret_access_key=sj4K8Ufaf6Fs4uRV3tTYXngSrLISwvTy
4 aws_session_token=FwoGZXIvYXdzEKX//////////wEaD0mtJ2WmDVqY8wyCGSK9Ad8ssTQxMpSwJ
5
6
7
8
9
10
11
12
```

Below the code editor are tabs for DEBUG CONSOLE, TERMINAL, PROBLEMS, OUTPUT, and JUPYTER. The TERMINAL tab is active, showing a prompt: `○ ~/.aws $`. The bottom right corner contains terminal control icons.

Infrastructure as Code

Who are you?

- Get some basic info about your credentials and make sure everything is working

The screenshot shows a code editor interface with a terminal tab active. In the terminal, the command `aws sts get-caller-identity` is run, and the output shows the user's AWS credentials.

```
~/aws $ aws sts get-caller-identity
{
    "UserId": "AROAQ03ZKGV234PEGJHBZ:user2177621=Test_Student",
    "Account": "031929808245",
    "Arn": "arn:aws:sts::031929808245:assumed-role/voclabs/user2177621=Test_Student"
}
~/aws $
```

Infrastructure as Code

Who are you?

- Default output is JSON
- Can change to text or table

The screenshot shows a terminal window within a code editor interface. The terminal is executing two AWS CLI commands:

```
aws sts get-caller-identity --output table
```

Below the terminal, the output is displayed in a table format:

GetCallerIdentity	
Account	031929808245
Arn	arn:aws:sts::031929808245:assumed-role/voclabs/user2177621=Test_Student
UserId	AROAQ03ZKGV234PEGJHBZ:user2177621=Test_Student

The terminal prompt at the bottom is `~/.aws $`.

Infrastructure as Code

aws-cli

- The aws-cli is a command line interface to the core AWS API
- Everything you can do with the Web Console, you can do with the API and CLI

Infrastructure as Code

aws-cli

- If you've already created an EC2 instance, you have a security group already configured. Let's find it's ID

```
aws ec2 describe-security-groups --region us-east-1
```

Infrastructure as Code

aws-cli



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar with a file icon containing a '1' and a folder named 'DEMO'. The main area is a terminal window titled 'ec2.sh — Demo' containing the following script:

```
$ ec2.sh
1 #!/bin/bash -ex
2
3 aws run-instances \
4   --image-id ami-026b57f3c383c2eec \
5   --count 1 \
```

Below the terminal are tabs for DEBUG CONSOLE, TERMINAL, PROBLEMS, OUTPUT, and JUPYTER. The TERMINAL tab is selected. The output pane shows the results of running the AWS CLI command:

```
● ~/Demo $ aws ec2 describe-security-groups --region us-east-1 --output text
SECURITYGROUPS launch-wizard-1 created 2022-10-06T22:29:17.813Z
-wizard-1      031929808245      vpc-068199c168f5218af
IPPERMISSIONS  22      tcp      22
IPRANGES       0.0.0.0/0
IPPERMISSIONSEGRESS -1
IPRANGES       0.0.0.0/0
SECURITYGROUPS default VPC security group      sg-0d79cf666e4999710      default 031929808245      vpc-06
8199c168f5218af
IPPERMISSIONS -1
USERIDGROUPLPAIRS      sg-0d79cf666e4999710      031929808245
IPPERMISSIONSEGRESS -1
IPRANGES       0.0.0.0/0
○ ~/Demo $
```

A blue rectangular selection highlights the security group ID 'sg-05ec512166ea5e682' in the output. The status bar at the bottom right shows 'bash' and other terminal-related icons.

Infrastructure as Code

aws-cli

- Looking up information is fine, but can we make things?
- Let's deploy a new EC2 instance from the command line.

Infrastructure as Code

The screenshot shows the AWS Cloud9 IDE interface. The left sidebar has icons for EXPLORER, TERMINAL, PROBLEMS, OUTPUT, and JUPYTER. The terminal tab is active, showing the command `./ec2.sh` which runs the script `ec2.sh`. The script content is:

```
$ ec2.sh
1 #!/bin/bash -ex
2
3 aws ec2 run-instances \
4   --region us-east-1 \
5   --image-id ami-026b57f3c383c2eec \
6   --count 1 \
7   --instance-type t2.micro \
8   --associate-public-ip-address \
9   --security-group-ids sg-05ec512166ea5e682 \
10  --key-name vockey
11
```

The output of the command in the terminal shows the JSON response from AWS, with the `InstanceId` field highlighted in blue:

```
~/Demo $ ./ec2.sh
+ aws ec2 run-instances --region us-east-1 --image-id ami-026b57f3c383c2eec --count 1 --instance-type
t2.micro --associate-public-ip-address --security-group-ids sg-05ec512166ea5e682 --key-name vockey
{
  "Groups": [],
  "Instances": [
    {
      "AmiLaunchIndex": 0,
      "ImageId": "ami-026b57f3c383c2eec",
      "InstanceId": "i-01dde1494e3e1e163",
      "InstanceType": "t2.micro",
      "KeyName": "vockey",
```

Infrastructure as Code

aws

The screenshot shows the AWS EC2 Instances Management console. The main view displays two instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm
-	i-0fccddaf84276a1a	Running	t2.micro	2/2 checks passed	No alarm
<input checked="" type="checkbox"/>	i-01dde1494e3e1e163	Running	t2.micro	Initializing	No alarm

The instance **i-01dde1494e3e1e163** is selected and highlighted with a blue border. The details page for this instance is open, showing the following information:

Instance: i-01dde1494e3e1e163

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

Instance summary

Instance ID	Public IPv4 address	Private IPv4 addresses
i-01dde1494e3e1e163	184.72.74.136 open address ↗	172.31.30.228
IPv6 address	Instance state	Public IPv4 DNS
-	Running	

CloudFormation

AWS CloudFormation

Amazon's first party Infrastructure as Code service

- Refers to both the templating syntax as well as the AWS service
- Create text file templates which can be repeatedly deployed
- A deployment is called a “stack”

AWS CloudFormation

Amazon's first party Infrastructure as

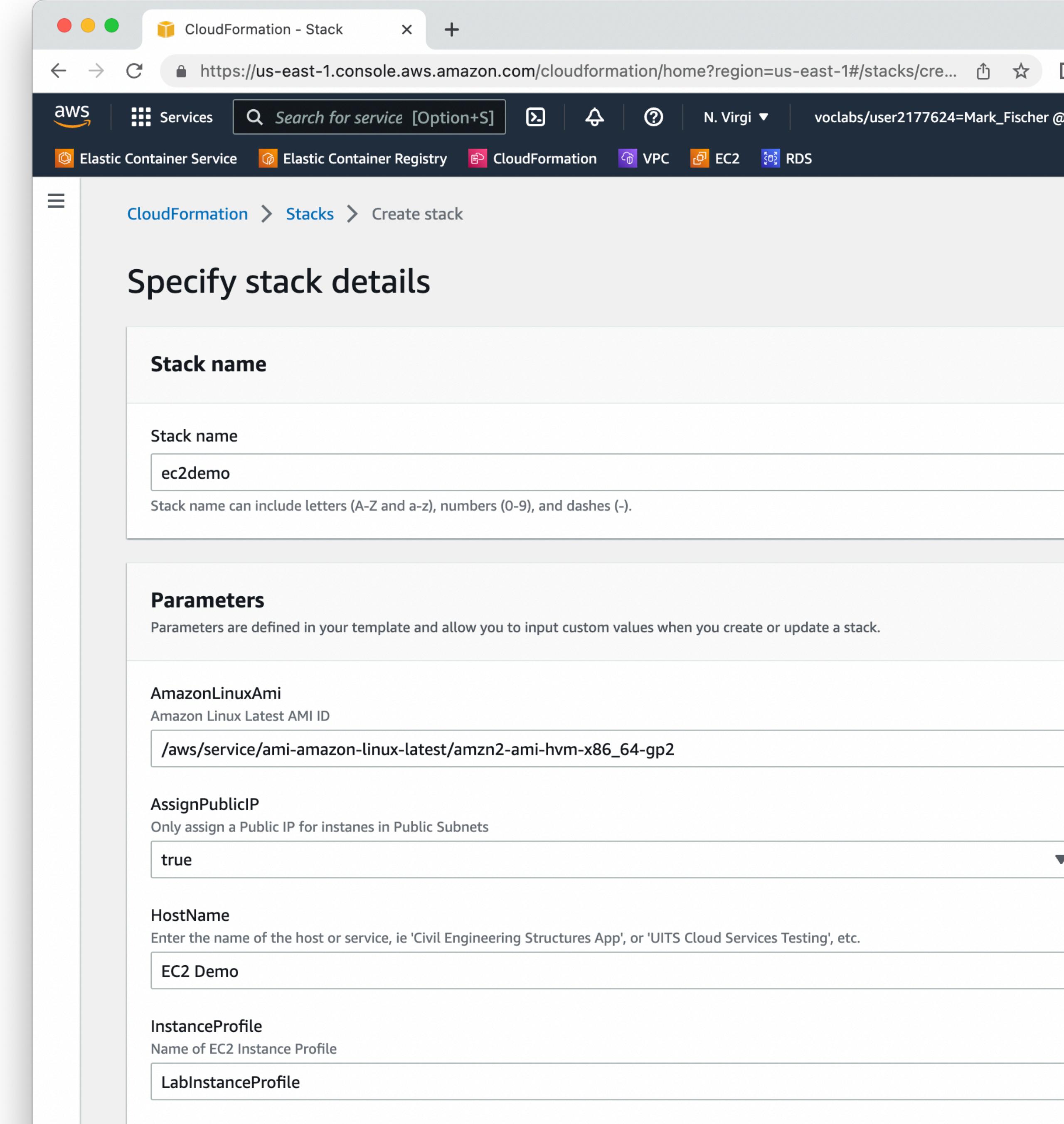
- Templates can be JSON or YAML formatted text files
- Top level sections: Parameters, Resources, Outputs and others
- Most data is basic key/value pairs
- YAML doesn't require you to quote every string

```
---  
# EC2 Basic CloudFormation Deployment  
# -----  
#  
# This CloudFormation template will deploy a single EC2  
# its own security group.  
  
AWSTemplateFormatVersion: "2010-09-09"  
  
Parameters:  
  HostName:  
    Type: String  
    Description: "Enter the name of the host or service,"  
  
Resources:  
  Ec2Instance:  
    Type: "AWS::EC2::Instance"  
    Properties:  
      ImageId: !Ref AmazonLinuxAmi  
      KeyName: !Ref KeyName  
      InstanceType: !Ref InstanceType  
      IamInstanceProfile: !Ref InstanceProfile  
  
  InstanceSecurityGroup:  
    Type: "AWS::EC2::SecurityGroup"  
    Properties:  
      GroupDescription: "Allow ssh to client host"  
      VpcId: !Ref VPCID  
      SecurityGroupIngress:  
        - IpProtocol: tcp  
          FromPort: 22  
          ToPort: 22  
          CidrIp: "0.0.0.0/0"  
  
Outputs:  
  InstancePublicIP:  
    Condition: AssignPublicIPCondition  
    Description: "The Public IP address of the instance"  
    Value: !GetAtt Ec2Instance.PublicIp
```

AWS CloudFormation

Infrastructure as Code service

- Templates can be uploaded to the AWS web console and deployed



AWS CloudFormation

Infrastructure as Code service

- Stack changes can be previewed before deployment to see what resources will be created or modified

The screenshot shows the AWS CloudFormation console interface. At the top, there's a navigation bar with tabs for Services, CloudFormation, VPC, EC2, and RDS. Below the navigation bar, the URL is https://us-east-1.console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/chan... . The main content area shows a stack named "ec2demo" with a change set ID "ec2demo-gpbgzjutig4-joqb6yubcvh". The status is "CREATE_COMPLETE" and the execution status is "AVAILABLE". The "Changes" tab is selected, showing two changes: one to add an EC2 instance and one to add an Instance Security Group. The "Template" tab is also visible.

CloudFormation > Stacks > ec2demo > Change sets: ec2demo-gpbgzjutig4-joqb6yubcvh

ec2demo-gpbgzjutig4-joqb6yubcvh

Delete Execute

Overview

Change set ID
arn:aws:cloudformation:us-east-1:561707296892:changeSet/ec2demo-gpbgzjutig4-joqb6yubcvh/cea9ed4c-e9e9-438a-8c7b-e4b515aa504d

Status
CREATE_COMPLETE

Description
-

Status reason
-

Created time
2022-10-23 15:20:02 UTC-0700

Execution status
AVAILABLE

Changes Input Template JSON changes Hook invocations

Changes (2)

Action Logical ID Physical ID Resource type

Action	Logical ID	Physical ID	Resource type
Add	Ec2Instance	-	AWS::EC2::Instance
Add	InstanceSecurityGroup	-	AWS::EC2::SecurityGroup

AWS CloudFormation

Infrastructure as Code service

- Can watch the progress of the stack deployment
- If anything fails, CloudFormation can either leave things in place and broken so you can examine things, or it can roll back all your changes

Timestamp	Logical ID	Status	Status reason
2022-10-23 15:22:30 UTC-0700	Ec2Instance	i CREATE_IN_PROGRESS	Resource creation in progress
2022-10-23 15:22:28 UTC-0700	Ec2Instance	i CREATE_IN_PROGRESS	-
2022-10-23 15:22:25 UTC-0700	InstanceSecurityGroup	✓ CREATE_COMPLETE	-
2022-10-23 15:22:24 UTC-0700	InstanceSecurityGroup	i CREATE_IN_PROGRESS	Resource creation in progress
2022-10-23 15:22:18 UTC-0700	InstanceSecurityGroup	i CREATE_IN_PROGRESS	-
2022-10-23 15:22:14 UTC-0700	ec2demo	i CREATE_IN_PROGRESS	User Initiated
2022-10-23 15:20:02 UTC-0700	ec2demo	i REVIEW_IN_PROGRESS	User Initiated

AWS CloudFormation

Infrastructure as Code service

- Stacks can be updated over time
- Stacks can be completely deleted when you're finished with it

ec2demo					
Stack info	Events	Resources	Outputs	Parameters	
Events (15)					
Search events					
Timestamp	Logical ID	Status	Status reason		
2022-10-23 15:25:09 UTC-0700	ec2demo	✓ DELETE_COMPLETE	-		
2022-10-23 15:25:08 UTC-0700	InstanceSecurityGroup	✓ DELETE_COMPLETE	-		
2022-10-23 15:25:06 UTC-0700	InstanceSecurityGroup	ℹ DELETE_IN_PROGRESS	-		
2022-10-23 15:25:05 UTC-0700	Ec2Instance	✓ DELETE_COMPLETE	-		
2022-10-23 15:24:34 UTC-0700	Ec2Instance	ℹ DELETE_IN_PROGRESS	-		
2022-10-23 15:24:31 UTC-0700	ec2demo	ℹ DELETE_IN_PROGRESS	User Initiated		
2022-10-23 15:23:05 UTC-0700	ec2demo	✓ CREATE_COMPLETE	-		
2022-10-23 15:23:03 UTC-0700	Ec2Instance	✓ CREATE_COMPLETE	-		

AWS Python SDK - boto3

AWS Language SDKs

Software Development Kit

- AWS Provides many ways to interact with its API
- RAW REST API
- AWS Web Console
- AWS CLI
- Programming Language SDKs

AWS Language SDKs

Programming Language SDKs

<https://aws.amazon.com/developer/tools/>

- Python
- JavaScript
- Node.js
- Java
- Go
- C++
- .NET
- Ruby
- Rust
- Swift

Python SDK - boto3

Authentication

- Just like the aws-cli, if you're making AWS API calls from outside of an AWS account, you need credentials
- The boto3 SDK knows to look for your [default] credentials from your ~/.aws/credentials file
- If you got the aws-cli working, then running python code from your laptop will also work
- If you want to run your python code inside of a container, you need to get credentials in to the container

Python SDK - boto3

Create an EC2 Instance

- The SDK documentation is essential

<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

Python SDK - boto3

Two SDK Models

- Each Service in the boto3 library presents two different interface models
 - client model
 - Closely maps directly to the AWS API itself / aws-cli
 - Returns dictionary mappings of the raw JSON responses
 - resource model
 - More object oriented
 - Returns python objects

Python SDK - boto3

Create an EC2 Instance

- We want the `boto3.client` for EC2 to start
- Documentation provides a comprehensive list of all the properties and methods available
- Many examples
- I almost always start here first, then go off to more broad searches if I need to

The screenshot shows a web browser displaying the Boto3 Docs 1.24.96 documentation for the EC2 Client. The URL is <https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ec2.html#client>. The page has a blue header bar with the title "Boto3 Docs 1.24.96 documentation". On the left, there is a sidebar titled "TABLE OF CONTENTS" with links to "Quickstart", "A sample tutorial", "Code examples", "Developer guide", "Security", and a "Available services" section containing a long list of AWS services. The "Available services" section is currently expanded. The main content area is titled "Client" and contains information about the `EC2.Client` class, which is described as a low-level client representing Amazon Elastic Compute Cloud (EC2). It explains that EC2 provides secure and resizable compute capacity in the cloud. The text continues to describe the benefits of using EC2, mentioning Amazon Virtual Private Cloud (Amazon VPC) and Amazon EBS. Below this, there is a section titled "To learn more, see the following resources:" with links to the Amazon EC2 product page and documentation, the Amazon EBS product page and documentation, the Amazon VPC product page and documentation, and the Amazon Web Services VPN product page and documentation. At the bottom of the main content area, there is a code snippet showing how to import the boto3 library and create an EC2 client:

```
import boto3
client = boto3.client('ec2')
```

Below the code snippet, it says "These are the available methods:" followed by a list of method names:

- `accept_reserved_instances_exchange_quote()`
- `accept_transit_gateway_multicast_domain_associations()`
- `accept_transit_gateway_peering_attachment()`
- `accept_transit_gateway_vpc_attachment()`
- `accept_vpc_endpoint_connections()`
- `accept_vpc_peering_connection()`
- `advertise_byoip_cidr()`
- `allocate_address()`
- `allocate_hosts()`
- `allocate_ipam_pool_cidr()`
- `apply_security_groups_to_client_vpn_target_network()`
- `assign_ipv6_addresses()`
- `assign_private_ip_addresses()`
- `associate_address()`
- `associate_client_vpn_target_network()`

Python SDK - boto3

Create an EC2 Instance

- Client version is `run_instances`
- Mostly matches the `aws-cli` but you can see similarities to the `CloudFormation` version as well
- Region is defined when creating the `client` object
- Requires more details for things like `NetworkInterfaces` and `Counts`

```
import boto3
from botocore.config import Config

conf = Config(region_name="us-east-1")
ec2 = boto3.client("ec2", config=conf)

call_result = ec2.run_instances(
    ImageId="ami-026b57f3c383c2eec",
    InstanceType="t3.micro",
    MinCount=1,
    MaxCount=1,
    KeyName="vokey",
    NetworkInterfaces=[
        {
            "DeviceIndex": 0,
            "SubnetId": "subnet-0cea5865199d0595c",
            "Groups": ["sg-07f090fb54ae76532"],
            "AssociatePublicIpAddress": True,
        }
    ],
)

print(call_result)
```

Python SDK - boto3

Create an EC2 Instance

- Response is a generic python dictionary with key/value pairs
- Useful if you only need cursory interaction with the resource after you create it

```
call_result["InstanceId"]
```

```
{'Groups': [], 'Instances': [{ 'AmiLaunchIndex': 0, 'ImageId': 'ami-026b57f3c383c2eec', 'InstanceId': 'i-0aadad17c8d49bf7a', 'InstanceType': 't2.micro', 'KeyName': 'rockey', 'LaunchTime': datetime.datetime(2022, 10, 23, 20, 45, 33, tzinfo=tzutc()), 'Monitoring': { 'State': 'disabled'}, 'Placement': { 'AvailabilityZone': 'us-east-1e', 'GroupName': '', 'Tenancy': 'default'}, 'PrivateDnsName': 'ip-172-31-63-12.ec2.internal', 'PrivateIpAddress': '172.31.63.12', 'ProductCodes': [], 'PublicDnsName': '', 'State': { 'Code': 0, 'Name': 'pending'}, 'StateTransitionReason': '', 'SubnetId': 'subnet-0cea5865199d', 'VpcId': 'vpc-0b1989c3c4cd0263a', 'Architecture': 'x86_64', 'BlockDeviceMappings': [], 'ClientToken': 'c259d26c-0056-41bb-a5b3cb42857d', 'EbsOptimized': False, 'EnaSupport': True, 'Hypervisor': 'xen', 'NetworkInterfaces': [ { 'Attachment': { 'AttachTime': datetime.datetime(2022, 10, 23, 20, 45, 33, tzinfo=tzutc()), 'AttachmentId': 'eni-attach-0d2727e02df2c2ea', 'DeleteOnTermination': True, 'DeviceIndex': 0, 'Status': 'att', 'NetworkCardIndex': 0}, 'Description': '', 'Groups': [ { 'Group': 'launch-wizard-1', 'GroupId': 'sg-07f090fb54ae76532'}], 'Ipv6Addresses': [], 'MacAddress': '06:3d:1a:e8:79:37', 'NetworkInterfaceId': 'eni-0a8b52f5531047feb', 'OwnerId': '561707296892', 'PrivateDnsName': 'ip-172-31-63-12.ec2.intern', 'PrivateIpAddress': '172.31.63.12', 'PrivateIpAddresses': [ { 'Primary': True, 'PrivateDnsName': 'ip-172-31-63-12.ec2.int', 'PrivateIpAddress': '172.31.63.12'}], 'SourceDestCheck': True, 'Status': 'in-use', 'SubnetId': 'subnet-0cea5865199d0595c', 'VpcId': 'vpc-0b1989c3c4cd0263a', 'InterfaceType': 'interface'}, 'RootDeviceName': '/dev/xvda', 'RootDeviceType': 'ebs', 'SecurityGroups': [ { 'GroupName': 'launch-wizard-1', 'GroupId': 'sg-07f090fb54ae76532'}], 'SourceDestCheck': True, 'StateReason': { 'Code': 'pending', 'Message': 'pending'}, 'VirtualizationType': 'hvm', 'CpuOptions': { 'CoreCount': 1, 'ThreadsPerCore': 1}, 'CapacityReservationSpecification': { 'CapacityReservationPreference': 'open'}, 'MetadataOptions': { 'State': 'pending', 'HttpTokens': 'optional'}}]}
```

Python SDK - boto3

Terminate an EC2 Instance

- The resource model allows us to manipulate objects
- Here we first create an EC2 instance object in our code
- Because it is a python object, we can easily inspect attributes and call methods

```
import boto3
from botocore.config import Config

conf = Config(region_name="us-east-1")
ec2 = boto3.resource("ec2", config=conf)
instance = ec2.Instance("i-0aafad17c8d49bf7a")

print(instance.state)
instance.terminate()
instance.wait_until_terminated()
print(instance.state)
```

```
$ python3 ec2-terminate.py
{'Code': 16, 'Name': 'running'}
{'Code': 48, 'Name': 'terminated'}
$
```

Terraform

Open-Source Multi-Provider Templating System

Terraform

Create an EC2 Instance

- Open-source tool spooned by HashiCorp
- Supports multiple cloud providers
- Has its own language that is similar to JSON, but supports comments, and built-in references and functions
- Install the terraform CLI tool

<https://www.terraform.io/downloads>

```
terraform {  
    required_providers {  
        aws = {  
            source  = "hashicorp/aws"  
            version = "~> 4.16"  
        }  
    }  
    required_version = ">= 1.2.0"  
}  
  
provider "aws" {  
    region = "us-east-1"  
}  
  
# Create a basic EC2 Instance  
resource "aws_instance" "app_server" {  
    ami                               = "ami-026b57f3c383c2eec"  
    instance_type                     = "t2.micro"  
    associate_public_ip_address      = true  
    subnet_id                         = "subnet-0cea5865199d059"  
    security_groups                   = ["sg-07f090fb54ae76532"]  
    key_name                          = "voockey"  
}
```

Comparison

So what should you use?

- “It depends”
- Each method presented here has advantages and disadvantages
- Significant overlap between tools
- Can always start simple with a shell script running aws-cli commands. As that becomes cumbersome move to either boto3 or CloudFormation/Terraform depending on needs

Version Control Systems

Basically git

Version Control Systems

It's just git these days

- A version control system aims to keep track of all the changes made to any of your project files
- Mostly focused on text files
 - Binary files can be versioned, but they are harder to look at differences
- If you're dealing with text files that might change, you should probably use a version control system

Version Control Systems

It's just git these days

- Years ago there used to be several competing version control systems
- These days the industry has basically settled on git
- Originally developed to manage the Linux kernel.
- Designed as a distributed version control system with direct peer-to-peer capabilities
 - Very rarely used in practice
- Hub & spoke model of older version control systems gave rise to GitHub
- GitHub ≠ git!

The git Version Control System

- A git repository is basically a folder with a hidden .git directory in it which contains state and history
- Files added to the folder can then be added to change sets and committed to the repository
- All of this can happen locally on your computer without needing a server
- If you want to use a service like GitHub, your local repository can be pushed to a remote repository hosted on GitHub.

git basics

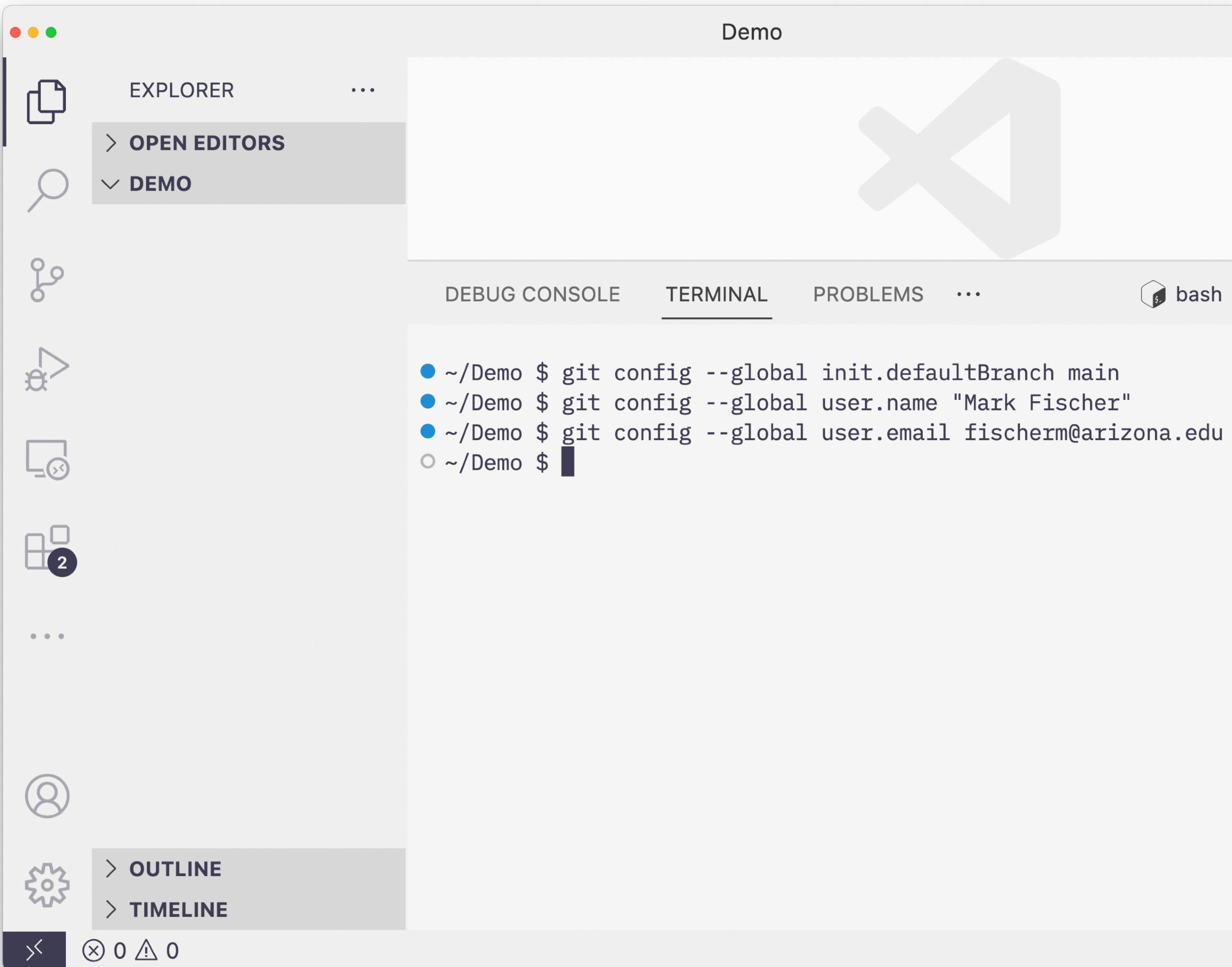
Setup

- <https://git-scm.com/downloads>
- Many platforms have git installed by default
 - macOS has git as part of Xcode
 - Windows installer
 - Linux package managers

git basics

Setup

- Initial setup commands
- Set your default branch name
- Set your user.name
- Set your user.email



The screenshot shows the VS Code interface with the title bar "Demo". The left sidebar has icons for Explorer, Search, and Open Editors (which is currently selected). The main area shows a terminal tab with the following commands:

```
~/Demo $ git config --global init.defaultBranch main
~/Demo $ git config --global user.name "Mark Fischer"
~/Demo $ git config --global user.email fischerm@arizona.edu
~/Demo $
```

git basics

Setup

- Create some files
- git init to initialize your current folder as a repository

The screenshot shows the VS Code interface with the title bar "Demo". The Explorer sidebar on the left shows a folder named "DEMO" containing "code.py" and "README.md". The Terminal tab is active, displaying the following output:

```
~/Demo $ git init
Initialized empty Git repository in /Users/fischerm/Demo/.git/
~/Demo $ ls -la
total 0
drwxr-xr-x  5 fischerm  staff   160 Oct 23 15:57 .
drwxr-xr-x+ 176 fischerm  staff  5632 Oct 23 15:53 ..
drwxr-xr-x  9 fischerm  staff   288 Oct 23 15:56 .git
-rw-r--r--  1 fischerm  staff    0 Oct 23 15:57 README.md
-rw-r--r--  1 fischerm  staff    0 Oct 23 15:57 code.py
~/Demo $
```

A blue box highlights the first command, "git init".

git basics

Setup

- Use `git status` to show what changes are not in your repository

The screenshot shows the VS Code interface with the title bar "Demo". The Explorer sidebar on the left lists a "DEMO" folder containing "code.py" and "README.md". The Terminal tab is active, displaying the output of a "git status" command:

```
total 0
drwxr-xr-x    5 fischerm  staff   160 Oct 23 15:57 .
drwxr-xr-x+ 176 fischerm  staff  5632 Oct 23 15:53 ..
drwxr-xr-x    9 fischerm  staff   288 Oct 23 15:56 .git
-rw-r--r--    1 fischerm  staff     0 Oct 23 15:57 README.md
-rw-r--r--    1 fischerm  staff     0 Oct 23 15:57 code.py

~/Demo $ git status
On branch main
```

Below the terminal, the status bar shows "main*" and other icons.

git basics

Setup

- Use `git add` to stage new or changed files

The screenshot shows the VS Code interface with the following elements:

- EXPLORER** pane on the left: Shows a folder named "DEMO" containing "code.py" and "README.md". Both files have an "A" icon next to them, indicating they are untracked.
- TERMINAL** tab is active at the bottom of the interface.
- Terminal Output:**
 - Untracked files:
(use "`git add <file>...`" to include in what will be committed)
 README.md
 code.py
 - nothing added to commit but untracked files present (use "`git add README.md code.py`")
 - ~/Demo \$ git add README.md code.py
 - ~/Demo \$ git status
 - On branch main
 - No commits yet
 - Changes to be committed:
(use "`git rm --cached <file>...`" to unstage)
 new file: README.md
 new file: code.py
 - ~/Demo \$
- PROBLEMS** tab is also visible at the bottom.

git basics

Setup

- Use `git commit` to commit all staged changes to the repository along with a change log message
- Message can be provided inline with the `-m` option, or with a CLI text editor like `vim`

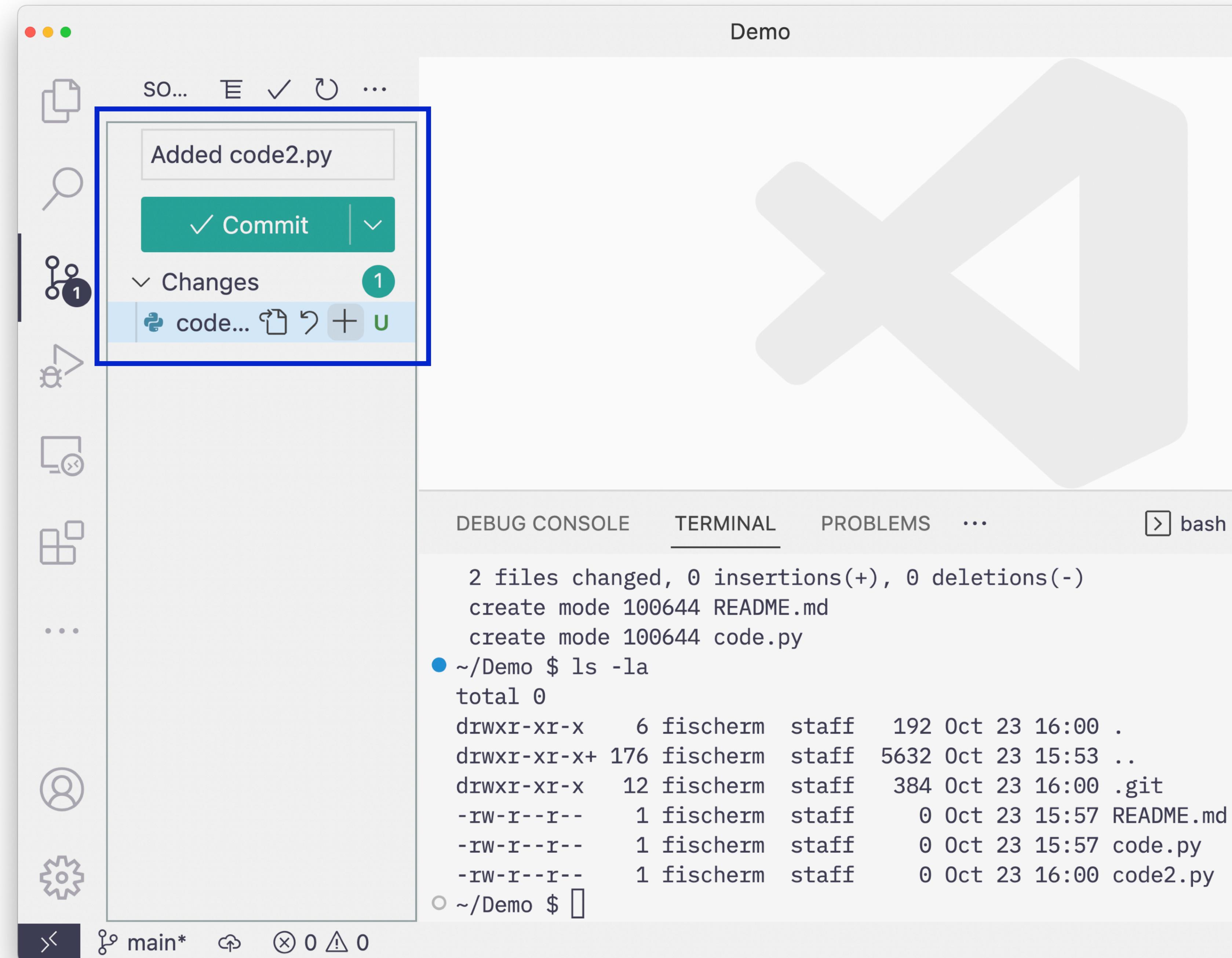
The screenshot shows the VS Code interface with the following details:

- EXPLORER** pane: Shows a folder named "DEMO" containing "code.py" and "README.md".
- TERMINAL** pane: Active tab, showing the output of a git session:
 - nothing added to commit but untracked files present (use "git add" to track changes)
 - ~/Demo \$ git add README.md code.py
 - ~/Demo \$ git status
 - On branch main
 - No commits yet
 - Changes to be committed:
(use "git rm --cached <file>..." to unstage)
 - new file: README.md
 - new file: code.py
 - ~/Demo \$ git commit -m "Initial commit"
[main (root-commit) ed7f28d] Initial commit
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md
create mode 100644 code.py
 - ~/Demo \$
- PROBLEMS** pane: Not visible.

git basics

Setup

- Tools like VS Code have built-in support for git
- Add and commit changed files directly in VS Code GUI



git basics

Setup

- Committing changes to files that are already tracked can be done with the `-a` option on the `git commit` command

The screenshot shows a VS Code interface with the following details:

- EXPLORER View:** Shows a folder named "DEMO" containing "code.py", "code2.py", and "README.md".
- Code Editor:** The file "code.py" is open, displaying Python code for creating an EC2 instance.
- Terminal View:** Shows the output of a git status command followed by a git commit command with the "-a" option.
- Status Bar:** Shows the file is at Line 4, Column 1, with 4 spaces, using UTF-8 encoding, and is a Python 3.9.13 64-bit file.

```
code.py — Demo

code.py ×
code.py > ...
1 import boto3
2
3 ec2 = boto3.client("ec2")
4
5 call_result = ec2.run_instances(
6     ImageId="ami-026b57f3c383c2eec",
7     InstanceType="t2.micro",
8     MinCount=1,
9     MaxCount=1,
```

```
DEBUG CONSOLE TERMINAL PROBLEMS ...
bash
~/Demo $ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
       modified:   code.py

no changes added to commit (use "git add" and/or "git commit"
~/Demo $ git commit -a -m "Added basic EC2 creation code"
[main cd274d6] Added basic EC2 creation code
  1 file changed, 21 insertions(+)
~/Demo $
```

Ln 4, Col 1 Spaces: 4 UTF-8 LF Python 3.9.13 64-bit ('3.9.13')

git basics

Setup

- VS Code also has built-in support for showing differences between files as you work

The screenshot shows the VS Code interface with a diff view between two versions of a file named `code.py`. The left pane shows the staged version (Working Tree) and the right pane shows the current version (Working Tree). The diff highlights changes in line 3, where the import statement has been updated from `boto3` to `botocore.config`.

```
code.py — Demo
```

```
code.py M code.py (Working Tree) M X
```

```
code.py > Config
```

```
1 import boto3 → 1 import boto3
2 → 2+ from botocore.config
3 → 4+ conf = Config(region_
4 → 5+ ec2 = boto3.client("ec2")
5 → 6
6 → 7 call_result = ec2.run_instances(
7 → 8 ImageId="ami-026b57f3c383c2eec",
8 → 9 InstanceType="t2.micro",
9 → 10 MinCount=1,
10 → 11 MaxCount=1,
11 → 12 KeyName="vokey",
12 → 13 NetworkInterfaces=[
13 → 14 }
```

DEBUG CONSOLE TERMINAL PROBLEMS OUTPUT JUPYTER

```
[main cd274d6] Added basic EC2 creation code
1 file changed, 21 insertions(+)
● ~/Demo $ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   code.py

no changes added to commit (use "git add" and/or "git commit -a")
○ ~/Demo $
```

Ln 2, Col 1 Spaces: 4 UTF-8 LF Python 3.9.13 64-bit

git basics

Setup

- Can see a history of commits with the `git log` command
- Also shows up in the VS Code Timeline pane

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows a tree view with `DEMO` expanded, containing `code.py`, `code2.py`, and `README.md`.
- OUTLINE** sidebar: Shows a tree view with `code.py` expanded, containing three commit entries:
 - Add region info
 - Added basic EC2 creation code
 - Initial commit
- code.py** editor tab: Displays Python code for creating EC2 instances.
- TERMINAL** tab: Displays the output of the `git log` command, showing four commits from Mark Fischer on Oct 23, 2022.
- Bottom status bar:** Shows file name (`code.py`), line count (Ln 6), column count (Col 1), spaces count (Spaces: 4), encoding (UTF-8), line ending (LF), language (Python), and version (3.9.13).