E6156 – Topics in SW Engineering (F23)
Cloud Computing
Lecture 11: GraphQL, CI/CD

Course Checkpoint

Status

This course will cover core concepts in cloud computing. Students will get practical experience with the concepts by implementing an application in a group project that uses the technology. Some of the technology concepts that the course covers are:

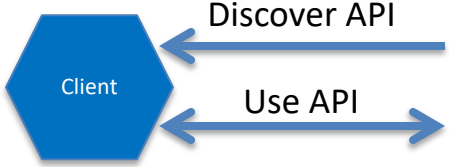
- REST
- Infrastructure-as-a-Service.
- Container-as-a-Service>
- Platform-as-a-Service.
- Software-as-a-Service.
- Database-as-a-Service.
- Function-as-a-Service.
- Backend-as-a-Service.
- Cloud Security.
- Multi-Tier Web Applications.
- Microservice.
- Web and cloud APIs.
- Event Driven Architectures.
- Service Orchestration.
- Systems and Application Management.
- Continuous Integration/Continuous Delivery
- Middleware
- Infrastructure as Code

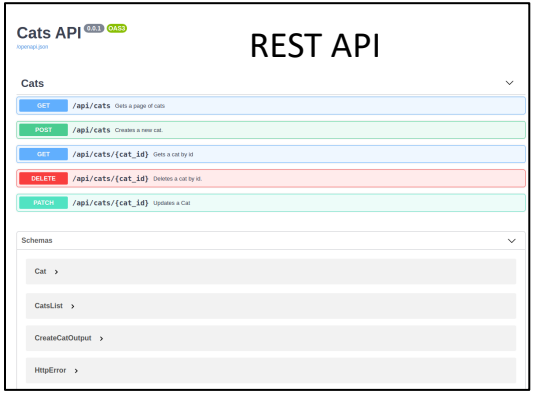
- We have “covered” most of the basic concepts.
- We should “focus” on project completion.
- I want to introduce four concepts:
  - GraphQL
  - Middleware
  - CI/CD
  - Infrastructure as Code
- Have your projects do something very simple.

GraphQL

Overview

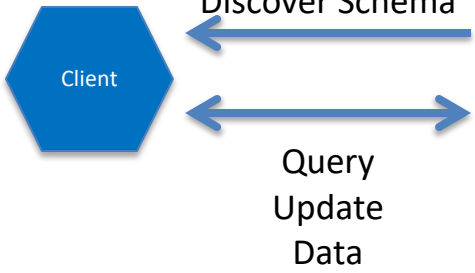
- REST
  - Serves document API
  - Client uses documented paths, methods, …
  - Resources (Data) are like a HTML document.

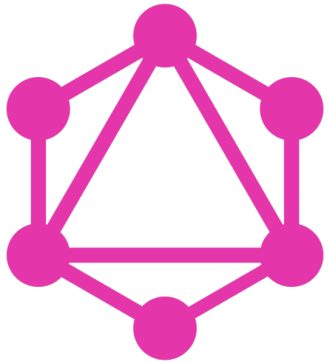




REST API

- GraphQL
  - Service documents resource model/schema
  - Client uses queries and updates (mutations).
  - Resources are like a (graph) database.



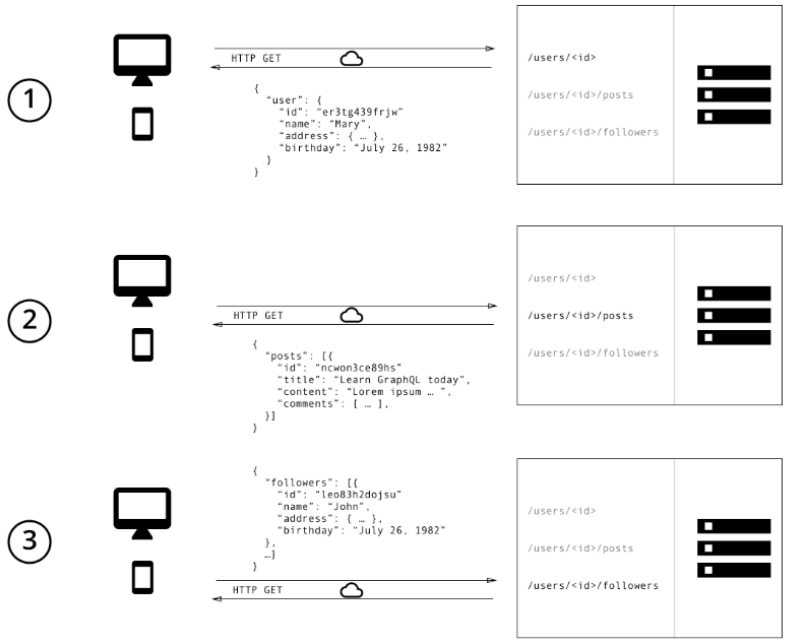


Resource Graph

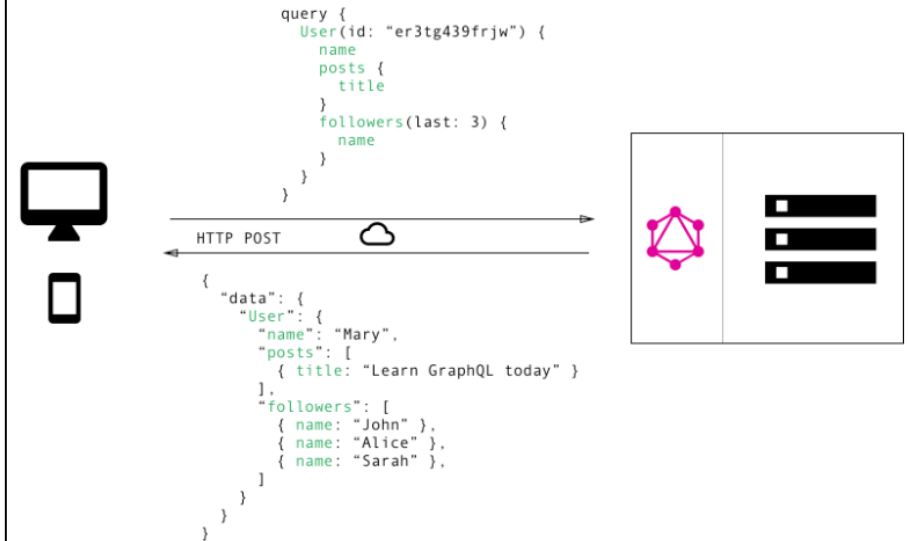
REST vs GraphQL (https://www.howtographql.com/basics/1-graphql-is-the-better-rest/)

Consider getting summary info from /users/{id}, /users/{id}/posts, /users/{id}/followers

With a REST API, you would typically gather the data by accessing multiple endpoints. In the example, these could be `/users/<id>` endpoint to fetch the initial user data. Secondly, there's likely to be a `/users/<id>/posts` endpoint that returns all the posts for a user. The third endpoint will then be the `/users/<id>/followers` that returns a list of followers per user.

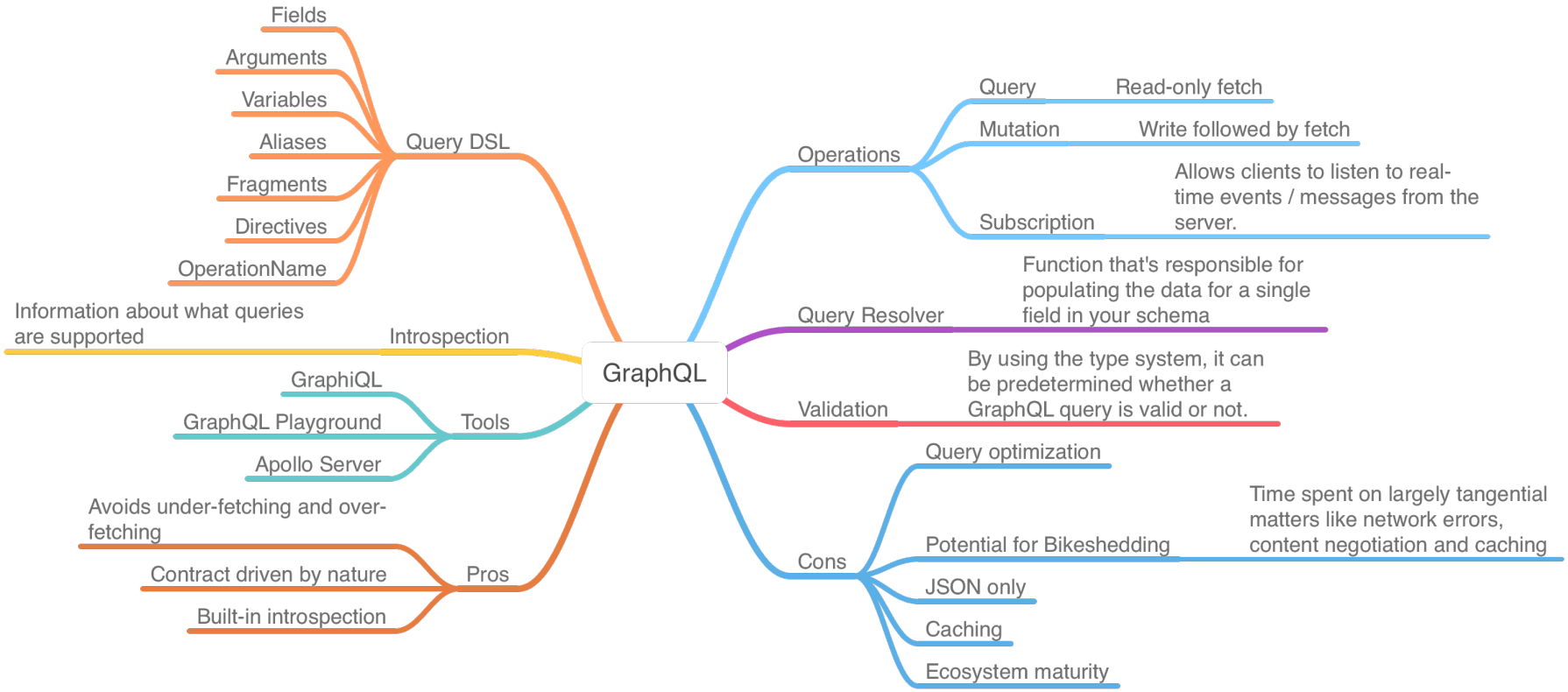


In GraphQL on the other hand, you'd simply send a single query to the GraphQL server that includes the concrete data requirements. The server then responds with a JSON object where these requirements are fulfilled.



Using GraphQL, the client can specify exactly the data it needs in a *query*. Notice that the *structure* of the server's response follows precisely the nested structure defined in the query.

Some GraphQL Concepts (https://www.peerislands.io/how-to-use-graphql-to-build-bffs/)



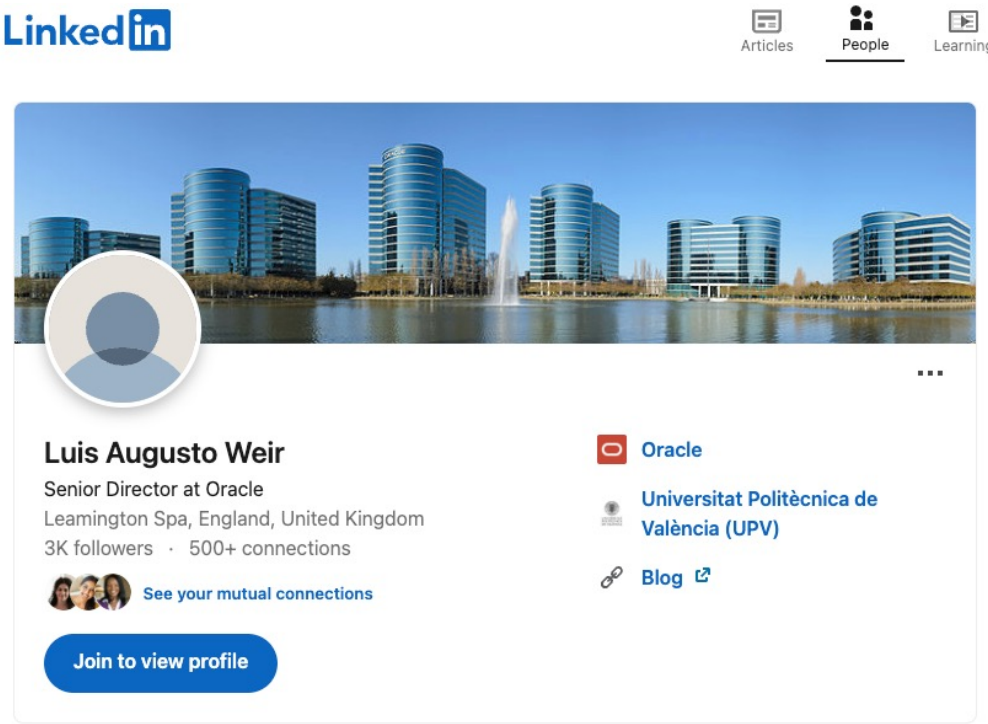
REST and GraphQL Summary

- Some resources:
  - A good, online playground with data: https://andybek.com/gql-nba
  - The Python GraphQL community on GitHub (https://github.com/graphql-python)
  - Go on GitHub (https://github.com/topics/graphql?l=go)

- This lecture uses Python and Graphene
  - https://docs.graphene-python.org/en/latest/quickstart/
  - FastAPI and GraphQL example:
    - https://github.com/danieldanuega/fastapi-graphql
    - Created a GitHub project from template.
      - Search tag on my hard drive is "#fastapi-graphql-1"
      - On GitHub: https://github.com/donald-f-ferguson/fastapi-graphql-example.git); used Docker compose version.
    - Created a Codespace from the templates.

Stolen Overview

https://uk.linkedin.com/in/lweir



Switch to PDF

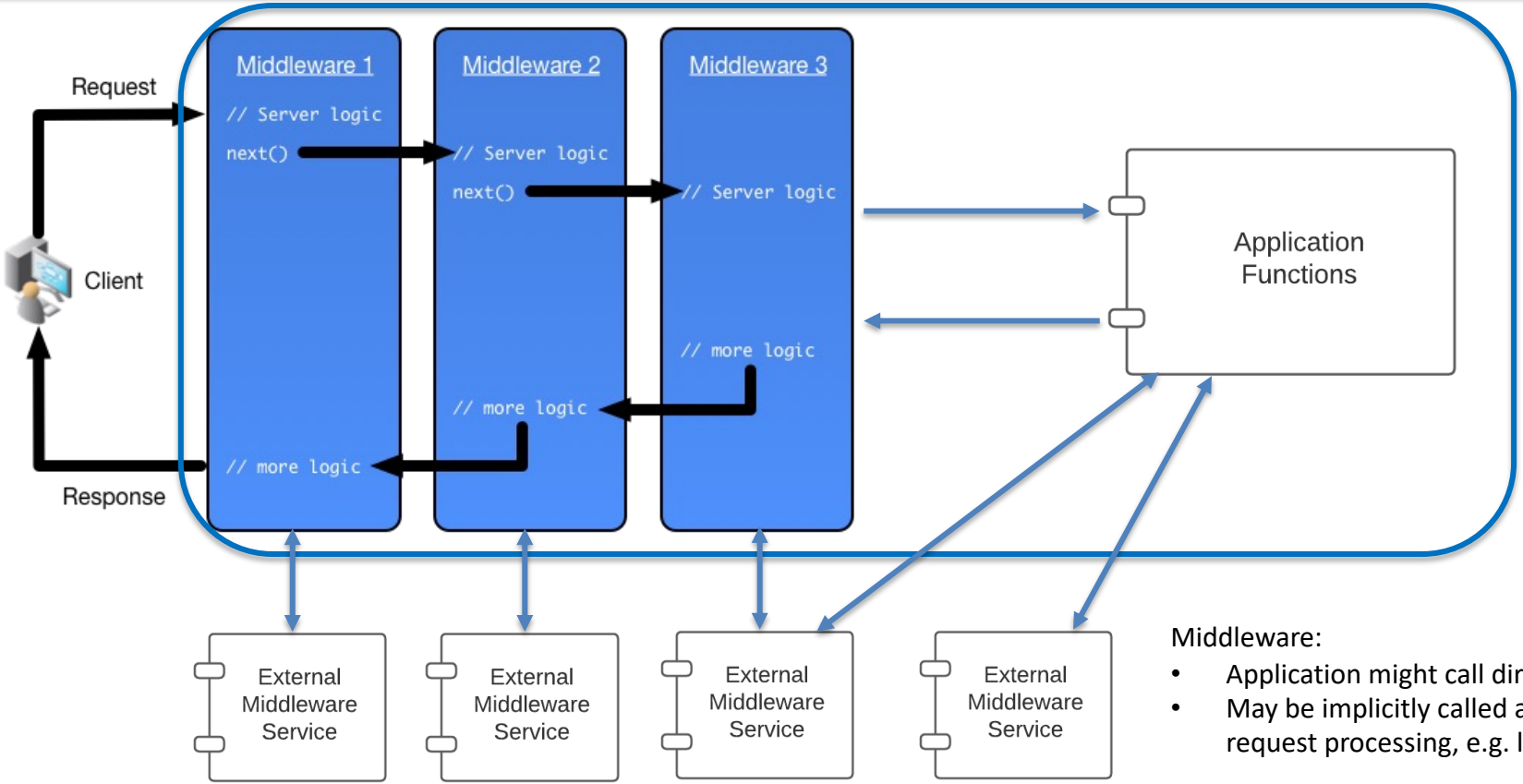
Middleware

What is middleware?

Middleware is software that lies between an operating system and the applications running on it. Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications. It's sometimes called plumbing, as it connects two applications together so data and databases can be easily passed between the "pipe." Using middleware allows users to perform such requests as submitting forms on a web browser, or allowing the web server to return dynamic web pages based on a user's profile.

Common middleware examples include database middleware, application server middleware, message-oriented middleware, web middleware, and transaction-processing monitors. Each program typically provides messaging services so that different applications can communicate using messaging frameworks like simple object access protocol (SOAP), web services, representational state transfer (REST), and JavaScript object notation (JSON). While all middleware performs communication functions, the type a company chooses to use will depend on what service is being used and what type of information needs to be communicated. This can include security authentication, transaction management, message queues, applications servers, web servers, and directories. Middleware can also be used for distributed processing with actions occurring in real time rather than sending data back and forth.

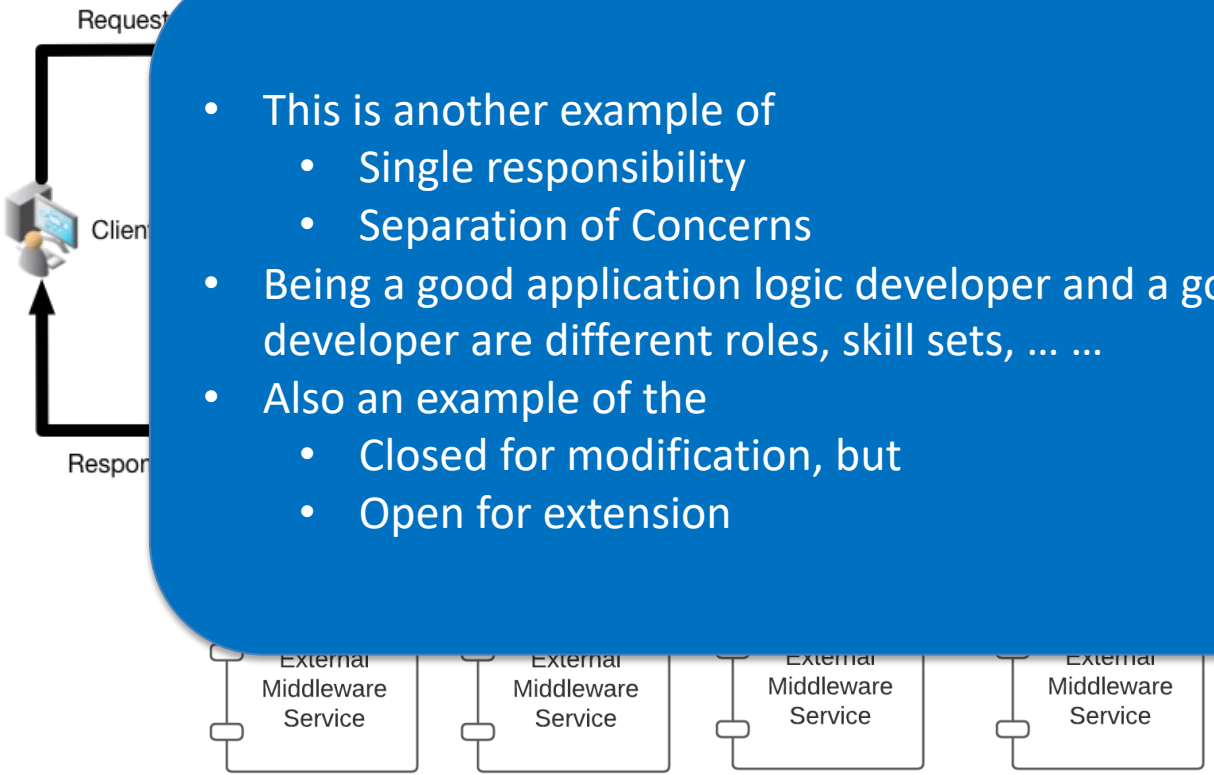
Middleware – Conceptual Model



Middleware:

- Application might call directly, e.g. DB
- May be implicitly called as part of request processing, e.g. logging.

Middleware – Conceptual Model



- This is another example of
  - Single responsibility
  - Separation of Concerns
- Being a good application logic developer and a good middleware developer are different roles, skill sets, … …
- Also an example of the
  - Closed for modification, but
  - Open for extension

- Application might call directly, e.g. DB
- May be implicitly called as part of request processing, e.g. logging.

FastAPI Middleware (https://medium.com/@saverio3107/mastering-middleware-in-fastapi-from-basic-implementation-to-route-based-

What Does Middleware Do?

1. **Incoming Request:** When a request comes in, the middleware can examine and even modify parts of the request like cookies, headers, query parameters, etc., before it reaches the actual API service method. For example, it can validate the request, authenticate the user, or log the request details.
2. **Outgoing Response**: After the API service method has processed the request and generated a response, the middleware gets a chance to inspect and modify the response before it goes back to the client. This can include things like changing the response headers, transforming the content, and so on.

FastAPI Middleware

Middleware

https://fastapi.tiangolo.com/tutorial/middleware/
https://fastapi.tiangolo.com/advanced/middleware/

You can add middleware to **FastAPI** applications.

A "middleware" is a function that works with every **request** before it is processed by any specific *path operation*. And also with every **response** before returning it.

- It takes each **request** that comes to your application.
- It can then do something to that **request** or run any needed code.
- Then it passes the **request** to be processed by the rest of the application (by some *path operation*).
- It then takes the **response** generated by the application (by some *path operation*).
- It can do something to that **response** or run any needed code.
- Then it returns the **response**.

Example

- Walk through Medium Article

- https://medium.com/@saverio3107/mastering-middleware-in-fastapi-from-basic-implementation-to-route-based-strategies-d62eff6b5463

- This is the very simple version.
  There is a more sophisticated approach to middleware in FastAPI.
  https://fastapi.tiangolo.com/advanced/middleware/

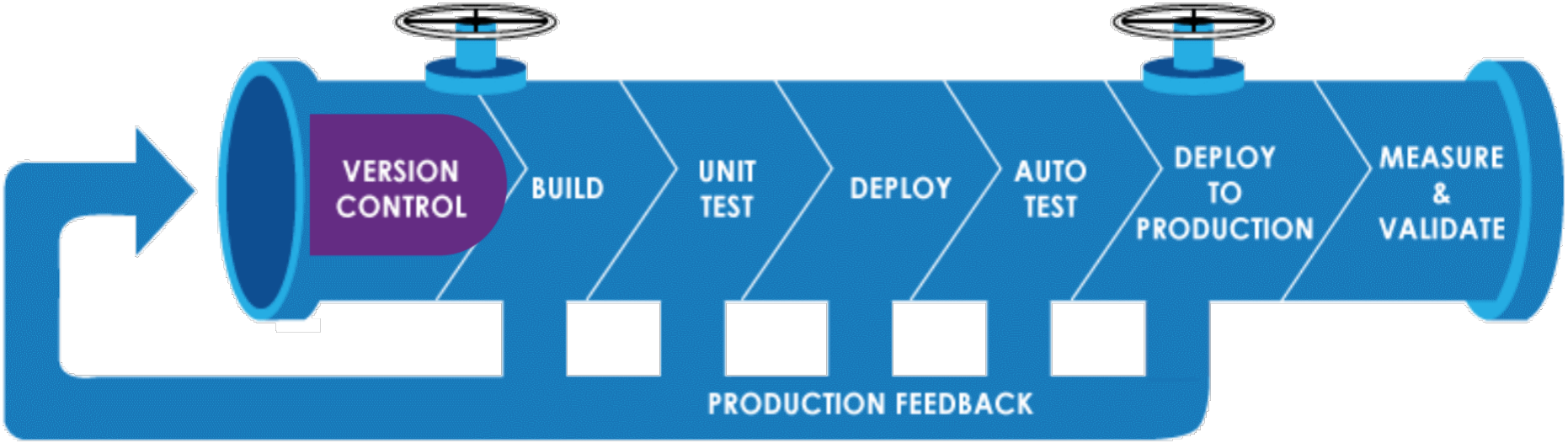
CI/CD

CI/CD (Part of DevOps)

Continuous delivery is a software development methodology where the release process is automated. Every software change is automatically built, tested, and deployed to production. Before the final push to production, a person, an automated test, or a business rule decides when the final push should occur. Although every successful software change can be immediately released to production with continuous delivery, not all changes need to be released right away.

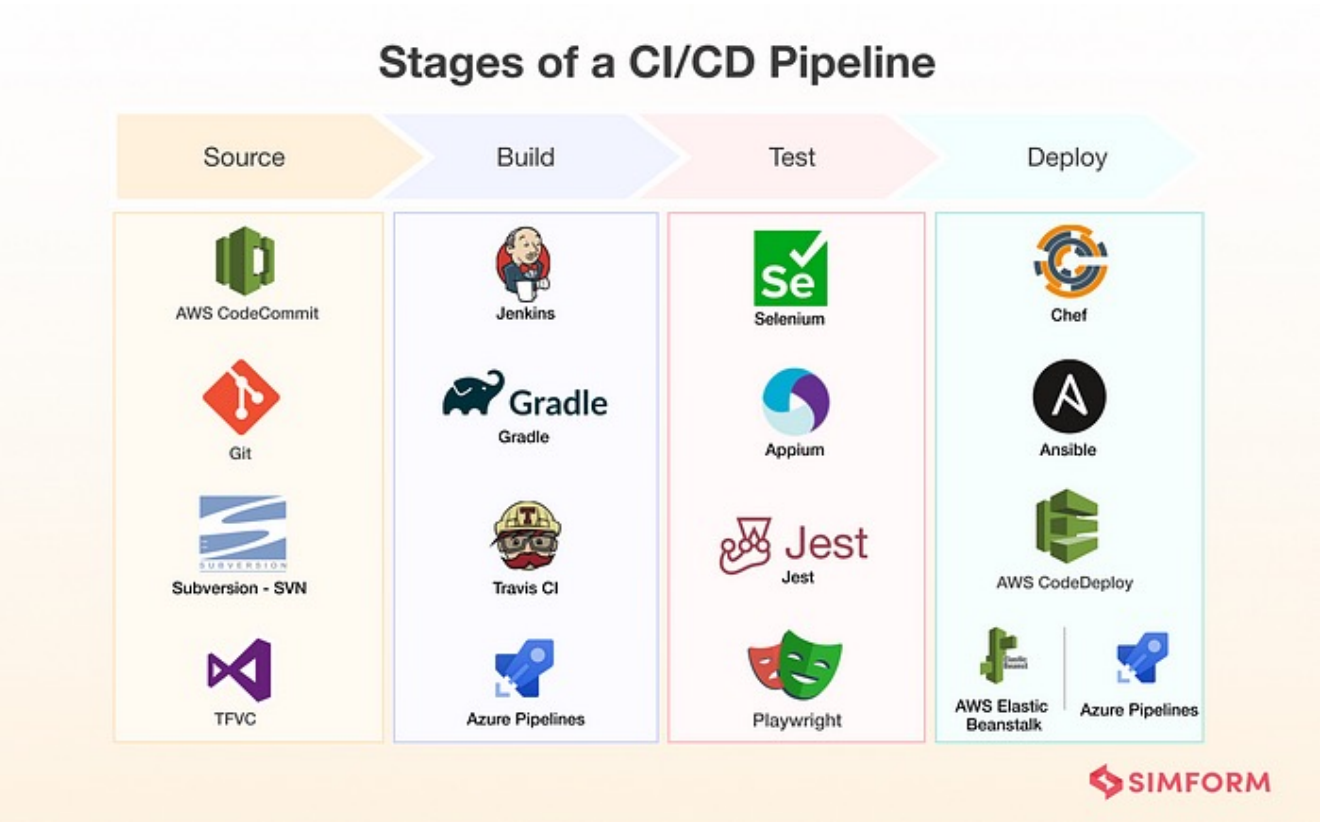
Continuous integration is a software development practice where members of a team use a version control system and frequently integrate their work to the same location, such as a main branch. Each change is built and verified to detect integration errors as quickly as possible. Continuous integration is focused on automatically building and testing code, as compared to *continuous delivery*, which automates the entire software release process up to production.

CI/CD Pipeline



- https://medium.com/jaanvi/basics-of-ci-cd-pipeline-5762e0eca44e

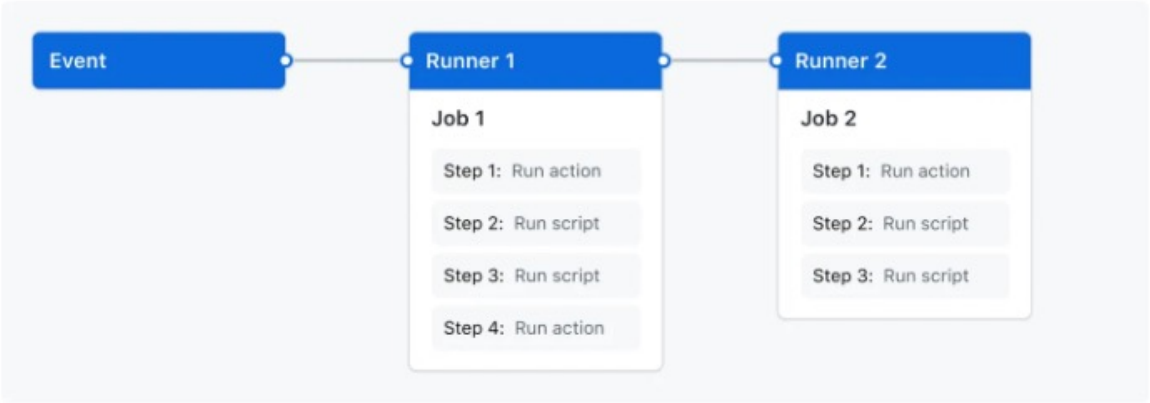
Some Technologies



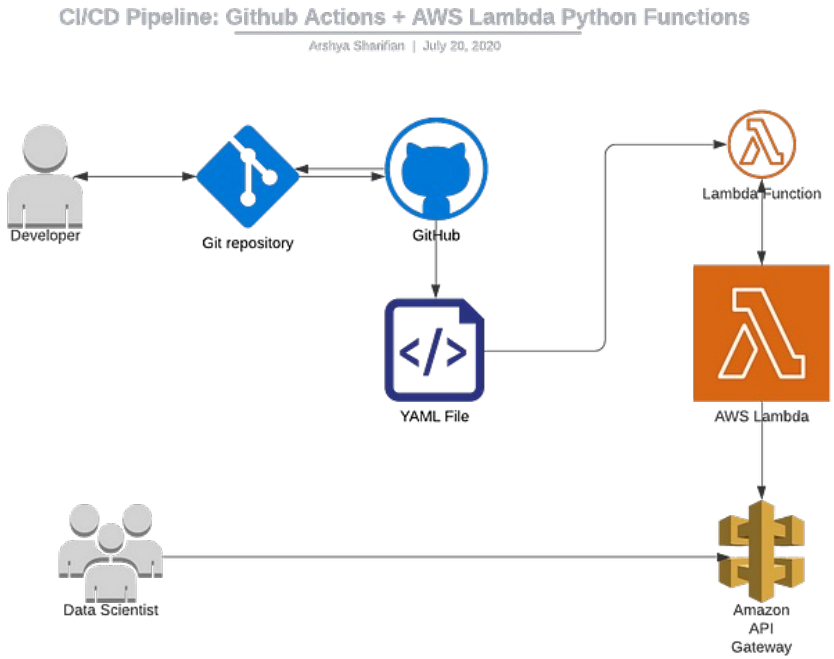
GitHub Actions

The components of GitHub Actions

You can configure a GitHub Actions *workflow* to be triggered when an *event* occurs in your repository, such as a pull request being opened or an issue being created. Your workflow contains one or more *jobs* which can run in sequential order or in parallel. Each job will run inside its own virtual machine *runner*, or inside a container, and has one or more *steps* that either run a script that you define or run an *action*, which is a reusable extension that can simplify your workflow.



Simple Example with GitHub Actions



https://towardsdatascience.com/modern-ci-cd-pipeline-git-actions-with-aws-lambda-serverless-python-functions-and-api-gateway-9ef20b3ef64a

Infrastructure as Code

Infrastructure as Code

- "Infrastructure as code (IaC) is the process of managing and provisioning computer data center resources through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools." (https://en.wikipedia.org/wiki/Infrastructure_as_code)

Tool	Released by	Method	Approach	Written in	Comments
CFEngine	Northern.tech (1993)	Pull	Declarative	C	-
Puppet	Puppet (2005)	Push and Pull	Declarative and imperative	C++ & Clojure since 4.0, Ruby	-
Chef	Chef (2009)	Pull	Declarative and imperative	Ruby	-
SaltStack	SaltStack (2011)	Push and Pull	Declarative and imperative	Python	-
Ansible / Ansible Tower	Red Hat (2012)	Push	Declarative and imperative	Python	-
Terraform	HashiCorp (2014)	Push	Declarative and imperative	Go	-
Otter	Inedo (2015)	Push	Declarative and imperative	-	Windows-oriented

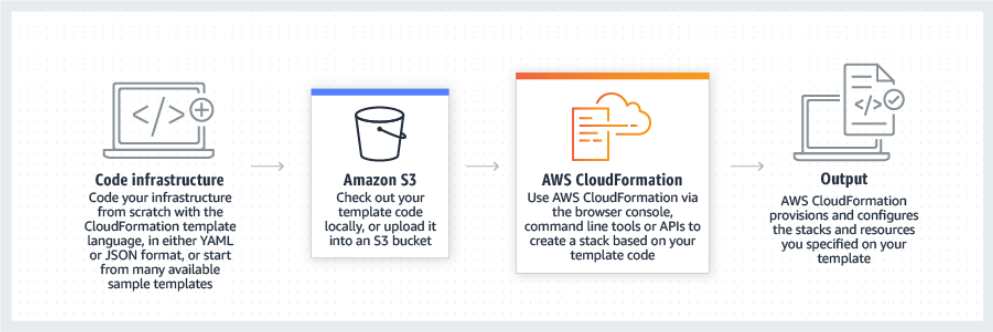
Code can be:

- Procedural/Declarative
- Push/Pull

CloudFormation

How it works

AWS CloudFormation lets you model, provision, and manage AWS and third-party resources by treating infrastructure as code.



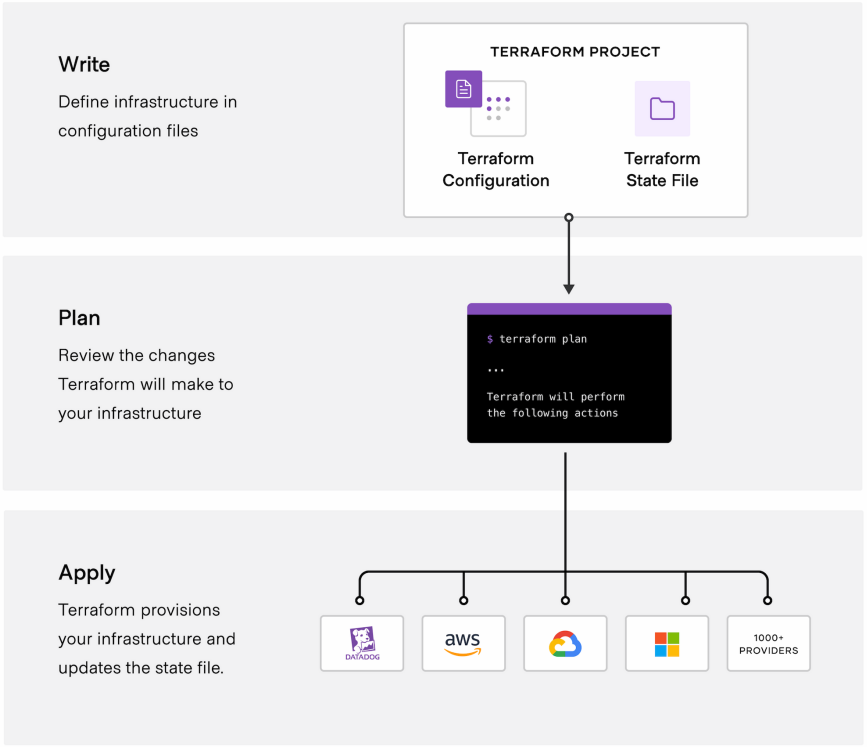
YAML

```yaml
Ec2Instance:
  Type: AWS::EC2::Instance
  Properties:
    KeyName: !Ref KeyName
    SecurityGroups:
      - !Ref Ec2SecurityGroup
    UserData:
      Fn::Base64:
        Fn::Join:
          - ":"
          - - "PORT=80"
            - "TOPIC="
            - !Ref MySNSTopic
    InstanceType: aa.size
    AvailabilityZone: aa-example-1a
    ImageId: ami-1234567890abcdef0
    Volumes:
      - VolumeId: !Ref MyVolumeResource
        Device: "/dev/sdk"
    Tags:
      - Key: Name
        Value: MyTag
```

- AWS CloudFormation simplifies provisioning and management on AWS. You can create templates for the service or application architectures you want and have AWS CloudFormation use those templates for quick and reliable provisioning of the services or applications (called “stacks”). You can also easily update or replicate the stacks as needed.
- One role (“admin”) creates templates that are well-designed, comply with policies, etc.
- “User” instantiation a template to produce infrastructure by specifying values for variables.

Terraform

https://developer.hashicorp.com/terraform/tutorials/configuration-language/resource#main-tf



Clone the example repository

Clone the Learn Terraform Resources repository, which contains example configuration to provision an AWS EC2 instance.

```
$ git clone https://github.com/hashicorp/learn-terraform-resources.git
```
Copy

Navigate to the repository directory in your terminal.

```
$ cd learn-terraform-resources
```
Copy

There are five files in this directory:

1. `init-script.sh` contains the provisioning script to install dependencies and start a sample PHP application
2. `terraform.tf` contains the `terraform` block that defines the providers required by your configuration
3. `main.tf` contains the configuration for an EC2 instance
4. `outputs.tf` contains the definitions for the output values of your resources
5. `README.md` describes the repository and its contents

Terraform Example

https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build

```
terraform {
 required_providers {
  aws = {
   source  = "hashicorp/aws"
   version = "~> 4.16"
  }
 }

 required_version = ">= 1.2.0"
}

provider "aws" {
 region  = "us-west-2"
}

resource "aws_instance" "app_server" {
 ami           = "ami-830c94e3"
 instance_type = "t2.micro"

 tags = {
  Name = "ExampleAppServerInstance"
 }
}
```

Terraform Block

The `terraform {}` block contains Terraform settings, including the required providers Terraform will use to provision your infrastructure. For each provider, the `source` attribute defines an optional hostname, a namespace, and the provider type. Terraform installs providers from the Terraform Registry by default. In this example configuration, the `aws` provider's source is defined as `hashicorp/aws`, which is shorthand for `registry.terraform.io/hashicorp/aws`.

Providers

The `provider` block configures the specified provider, in this case `aws`. A provider is a plugin that Terraform uses to create and manage your resources.

You can use multiple provider blocks in your Terraform configuration to manage resources from different providers. You can even use different providers together. For example, you could pass the IP address of your AWS EC2 instance to a monitoring resource from DataDog.

Resources

Use `resource` blocks to define components of your infrastructure. A resource might be a physical or virtual component such as an EC2 instance, or it can be a logical resource such as a Heroku application.

Resource blocks have two strings before the block: the resource type and the resource name. In this example, the resource type is `aws_instance` and the name is `app_server`. The prefix of the type maps to the name of the provider. In the example configuration, Terraform manages the `aws_instance` resource with the `aws` provider. Together, the resource type and resource name form a unique ID for the resource. For example, the ID for your EC2 instance is `aws_instance.app_server`.

Resource blocks contain arguments which you use to configure the resource. Arguments can include things like machine sizes, disk image names, or VPC IDs. Our providers reference lists the required and optional arguments for each resource. For your EC2 instance, the example configuration sets the AMI ID to an Ubuntu image, and the instance type to `t2.micro`, which qualifies for AWS' free tier. It also sets a tag to give the instance a name.