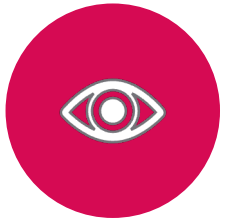
Microsoft Tech Days: Learn Go on Azure

Thursday 23 March 2023

Thank you for joining. The event will begin shortly after 14:00pm GMT.

In our event today we expect everyone to:



Be aware of others

Be friendly and patient

Be welcoming and respectful

Be open to all questions and viewpoints

Be understanding of differences

Be kind and considerate to others

For our full Code of Conduct or to report an issue, go to: aka.ms/eventsCoC


Ramadan
KAREEM

Meet your speakers

Adelina Simion

Technology Evangelist @ Form3

Liam Hampton

Senior Regional Cloud Advocate @ Microsoft

Agenda

Time	Topic
14:00 – 14:10	Welcome, introductions and agenda
14:10 – 15:00	The fundamental concepts within Go
15:00 – 15:30	Learn how Go integrates within Microsoft Azure
15:30 – 15:40	BREAK
15:40 – 16:40	Code with Liam and Adelina as they build a cloud hosted web application in Go
16:40 – 17:00	Q&A and wrap up/next steps

All timings are approximate.

Please submit questions throughout the event in the chat. Thanks!

Fundamental Go concepts

All the Go essentials we will be using today

Strong typing

- **Simplicity**, **safety** and **speed**
- At all points, we know the type of our variables, and what behaviour they expose.
- Go uses **type inference** to set the type of its variables based on their values. This makes it possible to write less verbose code.
- Dynamic typing is slower, and the compiler avoids runtime errors and fatal error cases due to undefined behaviours.
- The **fmt** package is part of the standard library and allows us to format and print strings.
- The Go **toolchain** builds and runs our programs. Runnable programs have a **main** function defined in a main package.

```
// Type inference in action

// x is of type int
x := 42

// y is of type string
y := "Hello, world!"

// v is of type []string
v := []string{"Liam", "Adelina"}
```

Functions

- Functions are declared using the **func** keyword.
- Go functions are natively supported and can be passed as: **variables**, **return types** and **parameters** for later invocation.
- **Anonymous functions** are also allowed.
- **Deferred** functions are useful for guaranteed clean up tasks.
- Function composition is a powerful tool that allows us to create extendable and idempotent code.

```
// Basic function
func hello() {
  fmt.Println("Hello, world!")
}

// Function as a variable
hello := func() {
 fmt.Println("Hello, world!")
}

// Invoke both declarations
hello()
```

Error handling

- Go functions can return multiple values.
- By convention, the error is the last returned value using the built-in **error** type.
- The zero value of the **error** type is **nil**.
- Errors should be handled first, keeping code minimally indented.

```
// error returning function
func hello() error{
  return errors.New(”my error msg”)
}

// handle error
if err:= hello(); err != nil {
  log.Fatal(err)
}
```

Custom types

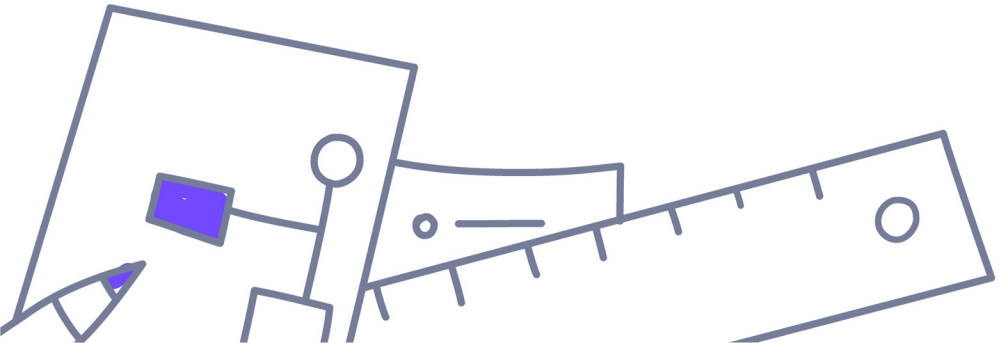
- Go is not an object-oriented language, as it does not support type hierarchy.
- **Structs** allow us to build custom types and behaviours.
- Unlike other languages, Go does not force the creation of constructors for custom types.
- They are a collection of fields, which can be partially initialised. Fields that are not initialised will assume the **zero value** of their type.
- Custom types can also define methods by using a special receiver argument that is the implicit first argument of the method.

```
// custom type
type person struct {
  name string
}

// constructor
func newPerson() person {
  return person{}
}

// invocation
p := newPerson()
```

Visibility

- Go code is organised in **packages**, which control the visibility of the **variables**, **types** and **functions** they contain.
- A folder may only contain a single package, but the package does not need to be named after the directory.
- Names can only be used once inside the same package.
- We can export fields outside their package by capitalising the first letter of their name.
- The **go.mod** file declares the module that all our packages are contained in.

```
.
├── Dockerfile
├── LICENSE
├── Makefile
├── README.md
├── service
│   └── helper.go
├── go.mod
├── go.sum
└── main.go
```



Interfaces

- Interfaces are collections of method signatures. They are declared with the **interface** keyword.
- They are **automatically implemented by the compiler** on types which satisfy the entire collection of methods.
- They are the primary way of implementing polymorphism in Go.
- Interfaces are often exported, while the structs remain visible only inside the package.

```
// declare interface
type User interface {
  Click()
}

// implement interface
type person struct {}

func (p person)Click() {
  fmt.Println("I've clicked a button!")
}
```

Goroutines

- Goroutines are known as lightweight threads. They are used to run functions concurrently inside our Go programs.
- We instruct the Go runtime to run a function in a new goroutine by using the **go** keyword.
- Starting a goroutine is non-blocking by design, otherwise we'd be running things sequentially.
- The program runs in its own goroutine, known as the main goroutine.
- The main goroutine has a parent child relationship with the goroutines it starts up.

```
// start goroutine
func main(){
  go func() {
    fmt.Println("Hello, from goroutine!")
  }()
}
```

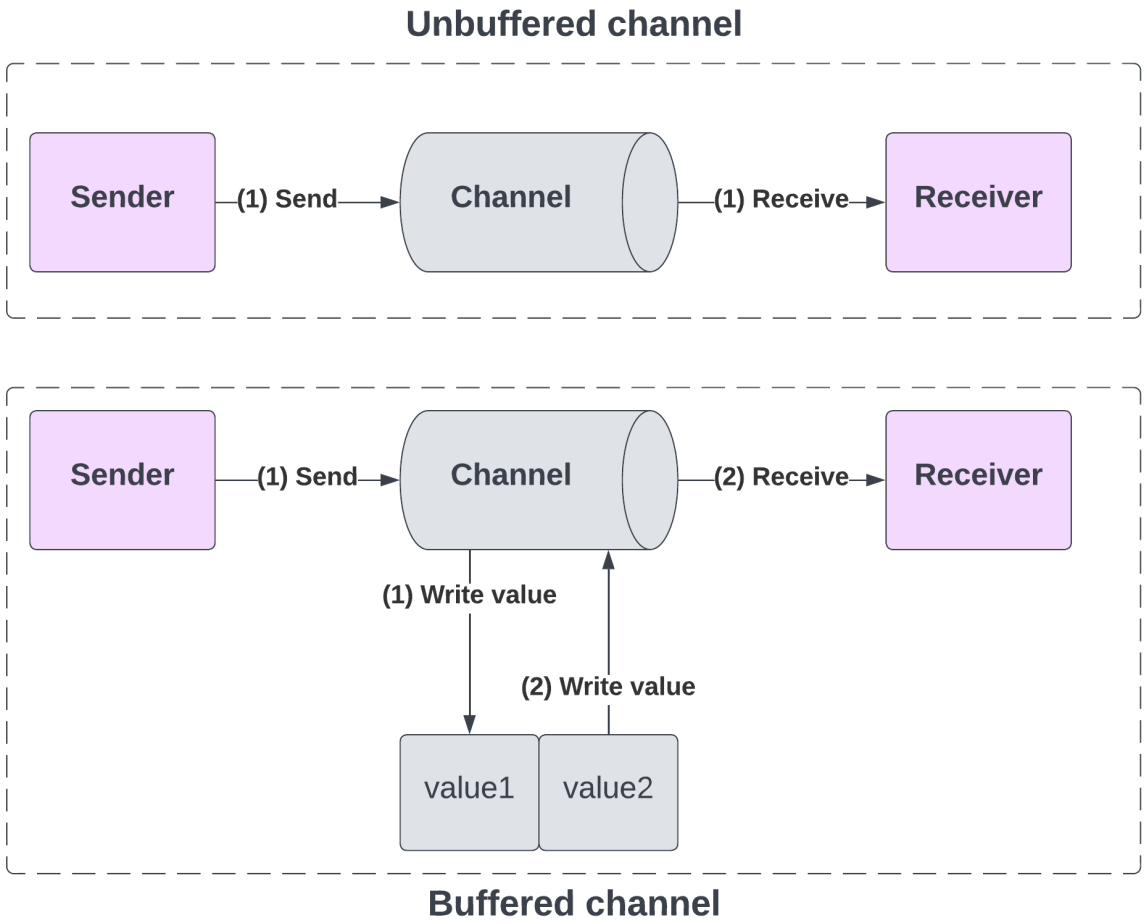
Channels

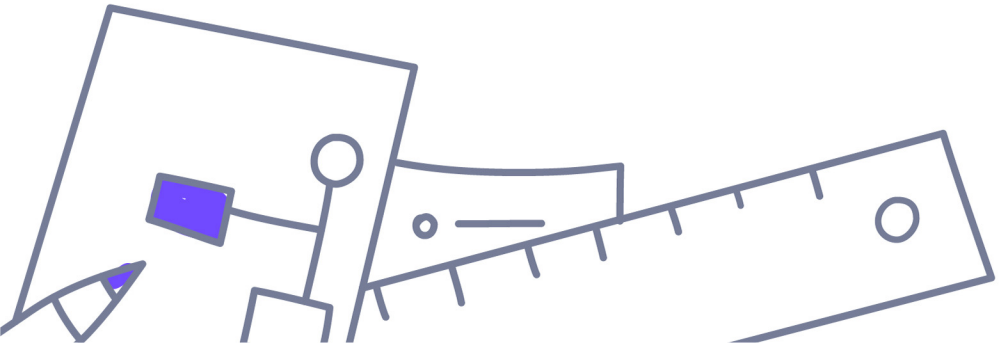
- It's discouraged to pass information between goroutines using shared memory variables.
- Channels are pipes which allow passing information in a thread-safe way.
- The type of variable that the channel supports is part of its initialisation.
- The send operation writes information through a channel, while the receive operation reads information from the channel.
- Sends and receives on a channel are blocking operations. They can be used for the synchronisation of goroutines.
- Messages are only read once.
- Once operations are completed, channels can be closed to signal to others that no more values will be sent through it.

```
// create channel
ch := make(chan string, 0)

// send to channel
go func() {
 ch <- "Hi!"
}()

// receive from channel
go func() {
 fmt.Println(<-ch)
}()
```

Buffered Channels

- By default, channels are unbuffered.
- They require both the sender and receiver to be available for the operation to be completed. These operations are synchronous.
- If one side is available without the other, then it will be blocked until the corresponding opposite operation is possible.
- Channels can be buffered with a pre-determined capacity to hold senders' values until receivers arrive.
- If there is space in the channel's queue, then the operation completes immediately.





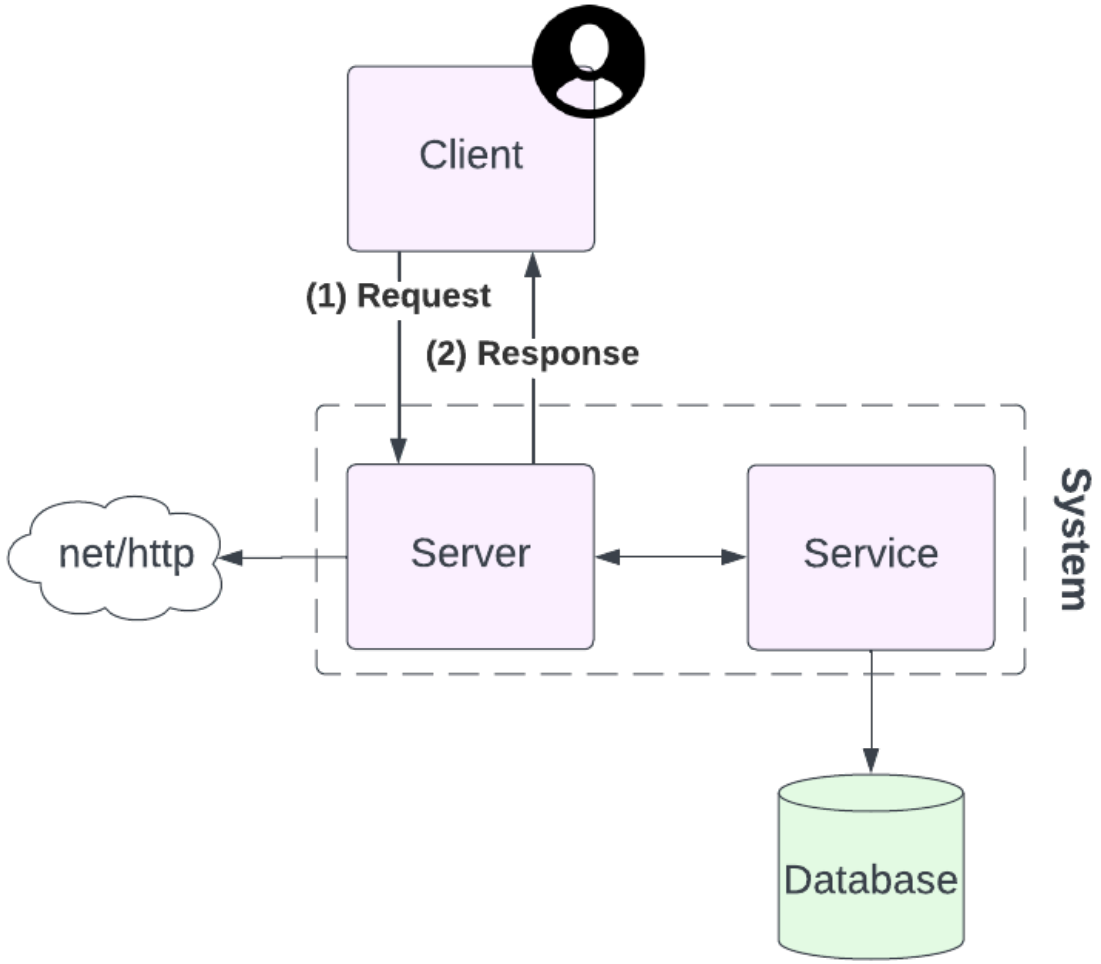
Unit testing

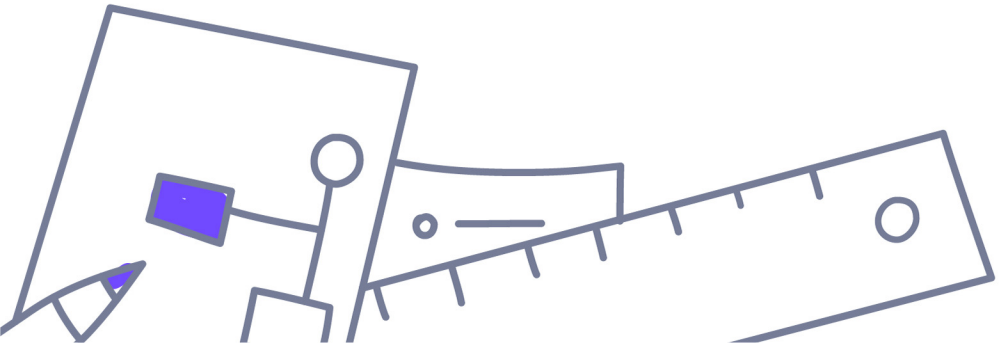
- Go's testing package allows us to write tests, verifications and benchmarks.
- Tests live alongside the code in **_test.go** files and have a set test signature.
- Coming from other languages, it might seem that Go's standard testing package is barebones.
- We can supplement it with other third-party libraries, but it's good to start with understanding how to write tests first.
- Testing concurrent code cannot prove the absence of bugs, but it can give us a statistical confidence of our code's behaviour under certain conditions.

```
// Test signature
func TestFunctionUnderTest(t *testing.T) {
  t.Run("subtest",func(t *testing.T){
    t.Fatal("test is failing")
  })
}

// Run the test
go test -run= TestFunctionUnderTest
go test ./...
```

The net/http package

- Go's HTTP package is easy to use and one of the reasons that Go is so widely used in web application development.
- Handlers respond to HTTP requests. Functions serving as handlers take in two parameters: **http.ResponseWriter** and **http.Request**.
- Handler functions are registered to a particular HTTP route using **http.HandleFunc**.
- The **net/http** package is responsible for passing the headers and requests to our custom registered handler functions.



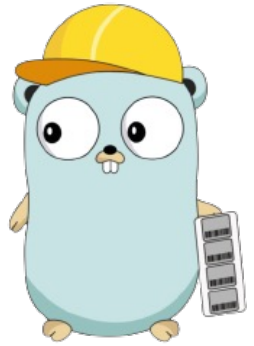


Go & Microsoft Azure

Learn how Go integrates within Microsoft Azure

Why use Go for cloud native projects?

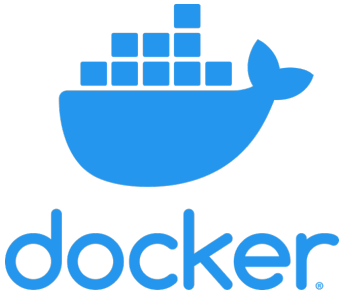
- Simplistic syntax
- Rich standard library
- Cross-platform
- Performance optimisation
- Scalability



Some of your favourite services are written in Go

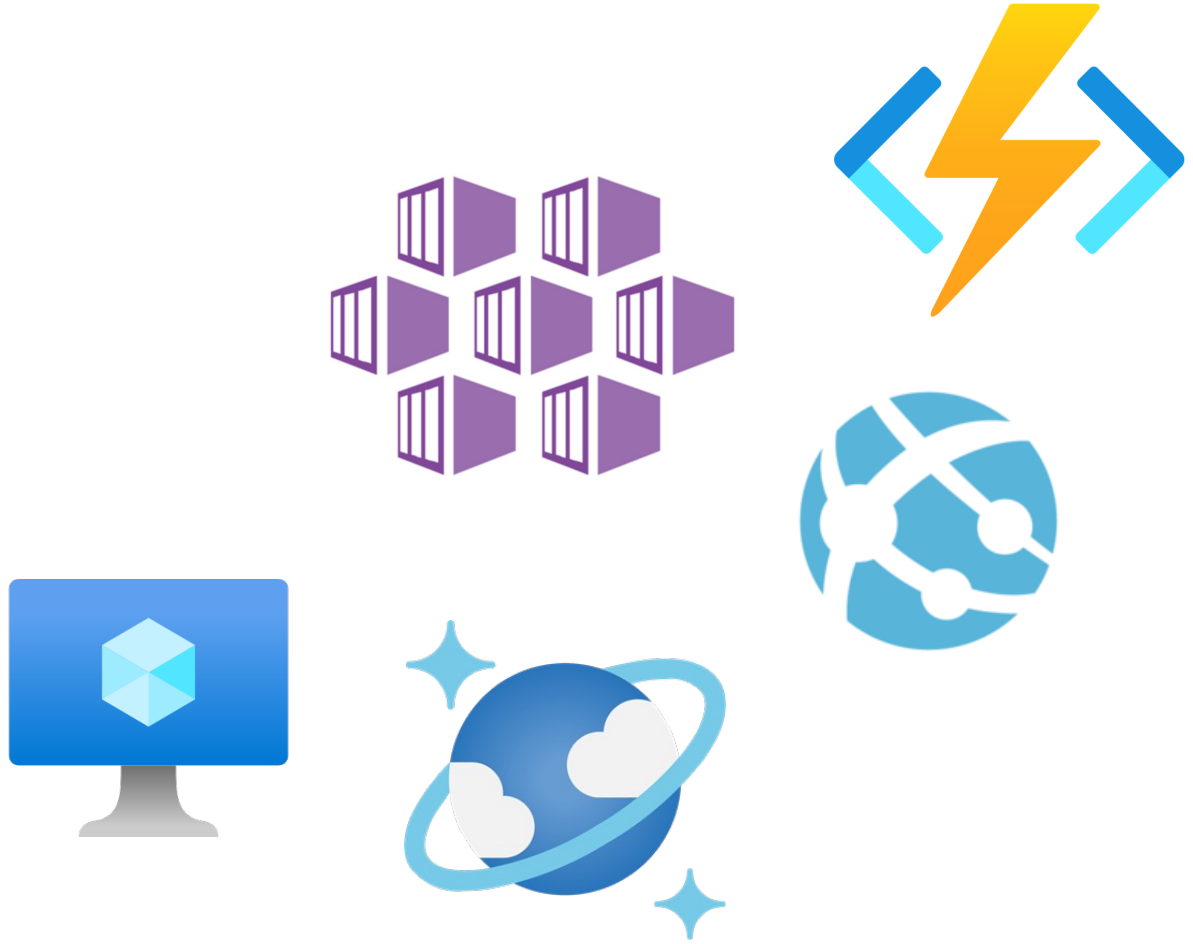
Azure Services
- Azure Go SDK
- Kiota Go SDK
- Azure Developer CLI
- Azure Kubernetes

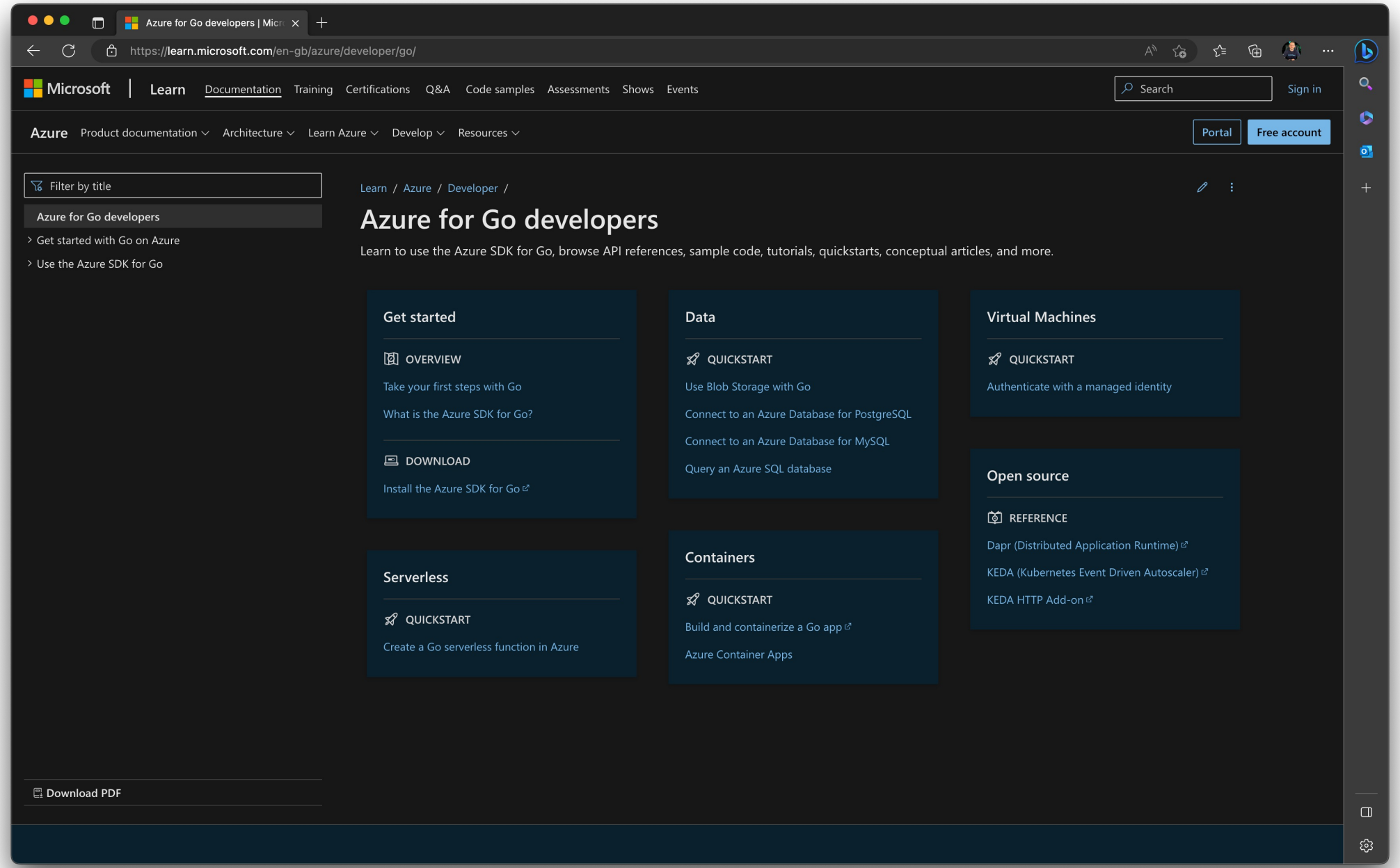
External projects
- Docker
- Dapr
- Virtual Kubelet (Azure Container Instances)



Services you can use with Go

- Managed Kubernetes (AKS)
- Virtual Machines
- CosmosDB
- Azure Database for MySQL and PostgreSQL
- Object Storage
- Azure App Services
- Azure Functions





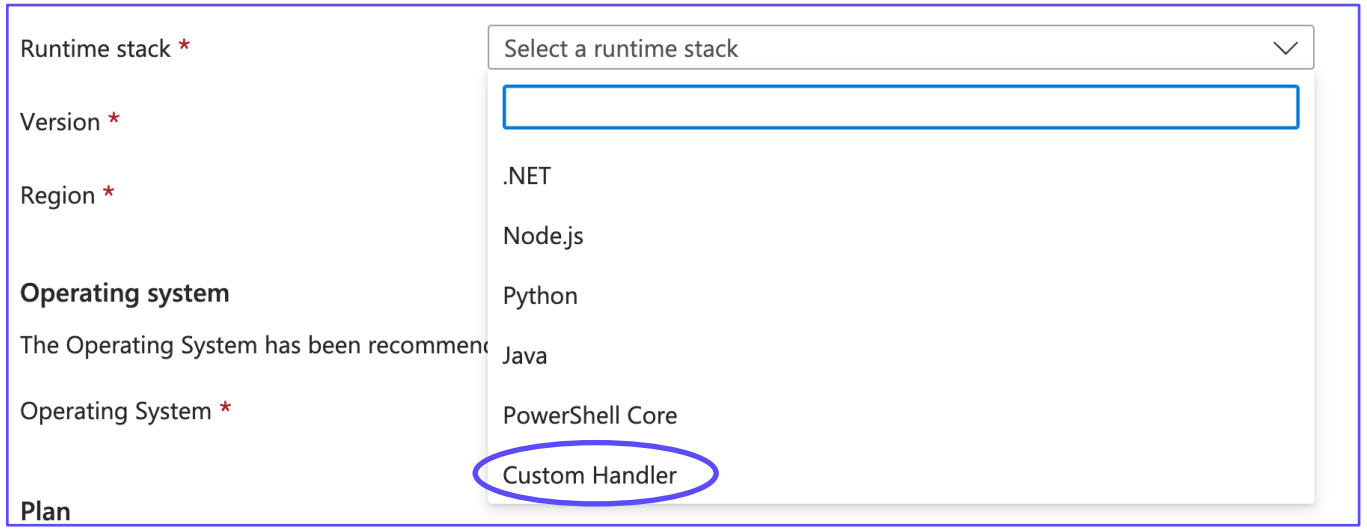
https://aka.ms/techdays/go/azure-services

Azure Functions

- Serverless compute platform for building and deploying event-driven applications
- Supports several different languages / runtime stacks
- Different deployment methods – GitHub, Zip deploy, FTP etc.
- Need to install Azure Functions Core Tools

Example use cases:

- Processing data from an IoT device
- Serverless API's
- Integrate with other Azure services

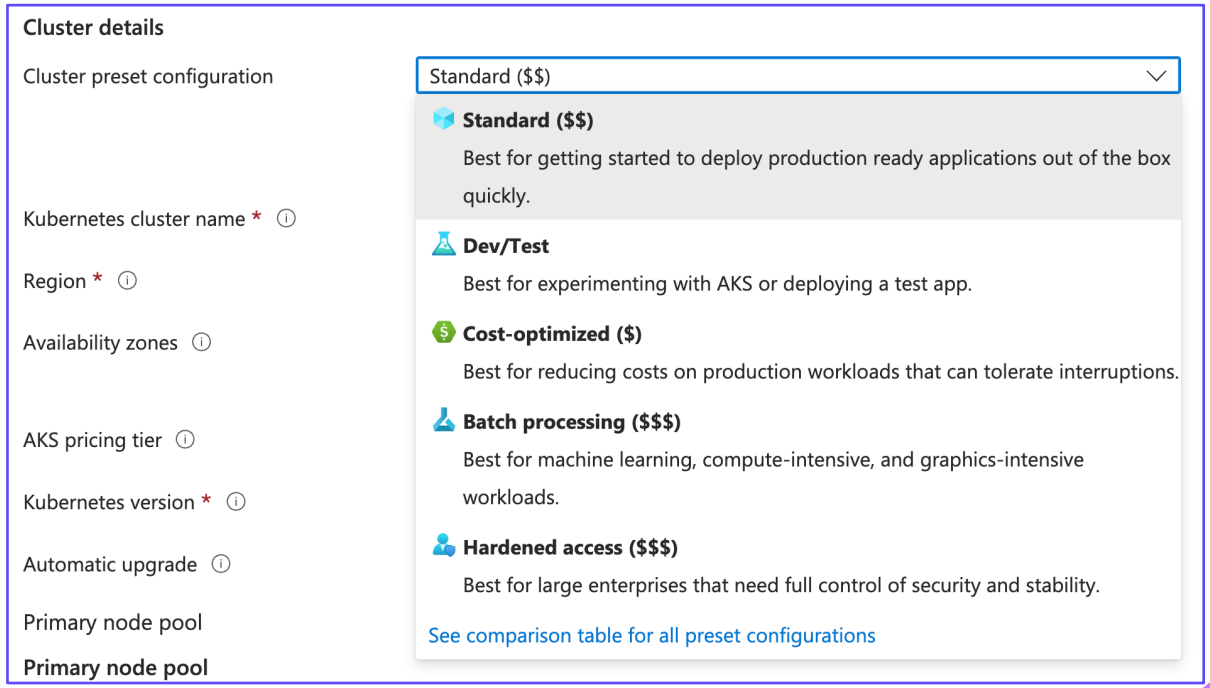


Azure Kubernetes Service (AKS)

- Managed container orchestration service offered by Azure
- Create scalable infrastructure with ease
- Simple deployment with containers
- Service discovery, load balancing and self healing
- Secret and config management

Example use cases:

- Building microservice architectures
- Building cloud native applications
- Serverless functions in AKS

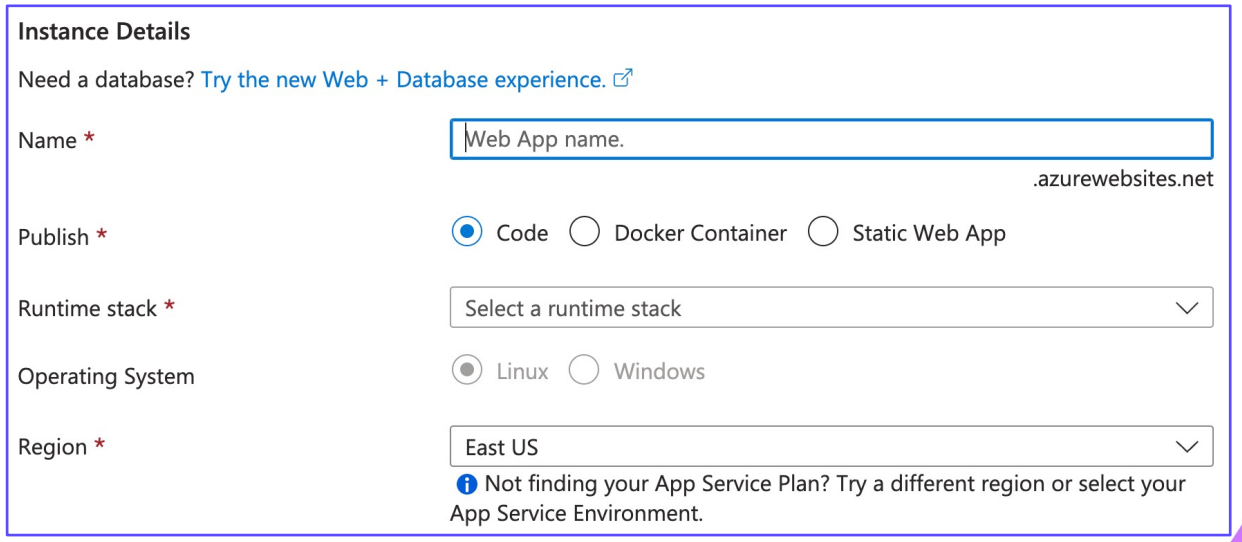


Azure App Service

- Fully managed serverless platform for building, scaling and deploying web applications and API's
- Great integration with other Azure services
- Autoscaling based on App Service Plan tier
- Streamlined deployment process

Example use cases:

- Building and deploying RESTful API's
- Hosting website / web apps
- Developing and deploying microservice infrastructure



Let’s get hands-on

GitHub repository:

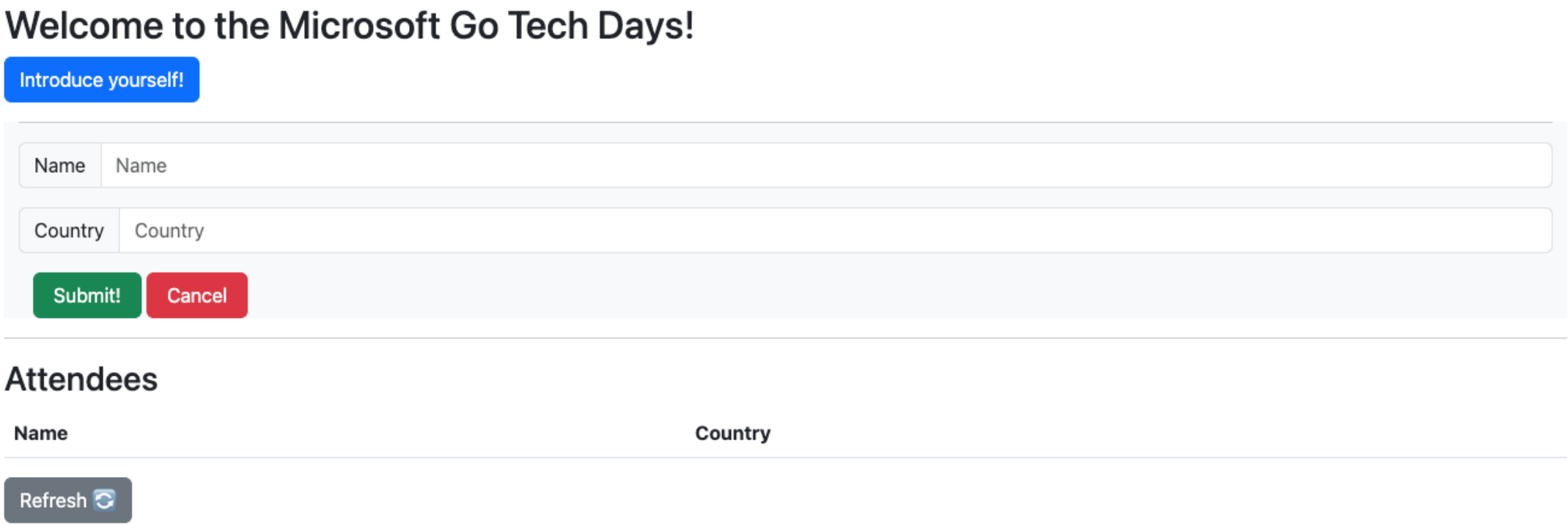
https://github.com/liamchampton/tech-days

Check it out during the break!

Code with Liam & Adelina

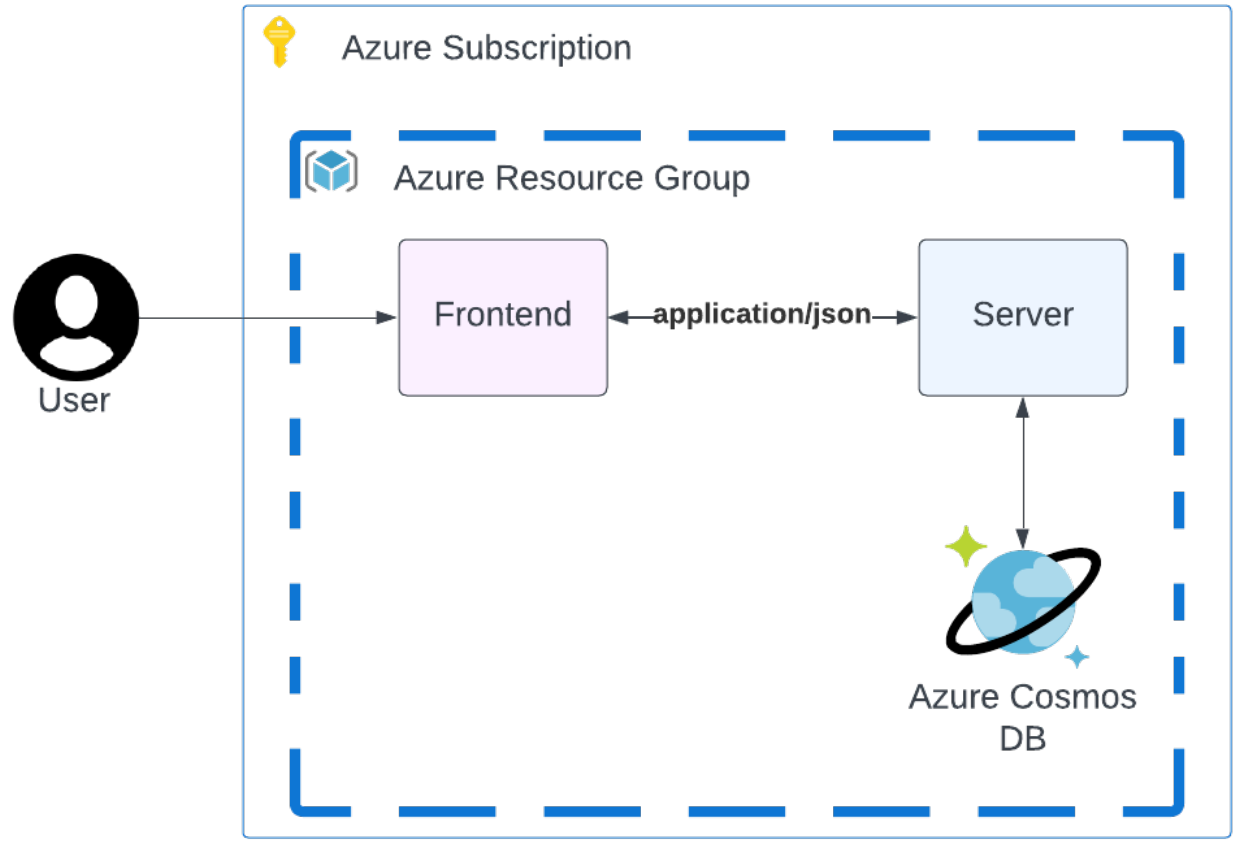
Let’s build something together!

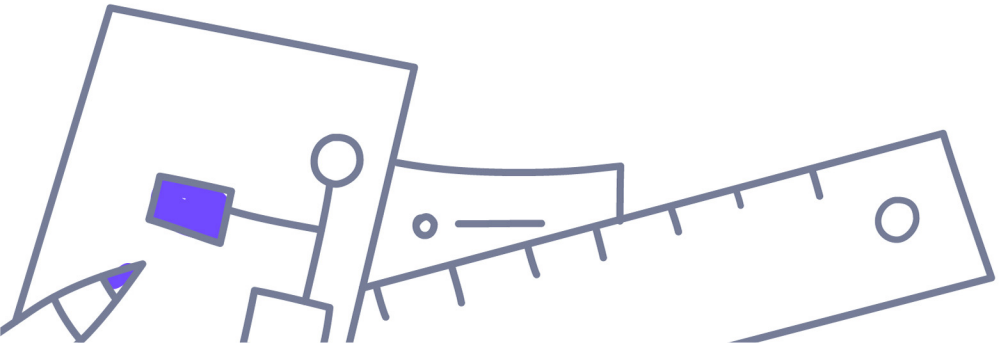
Our demo application will help us get to know each other!



System Overview

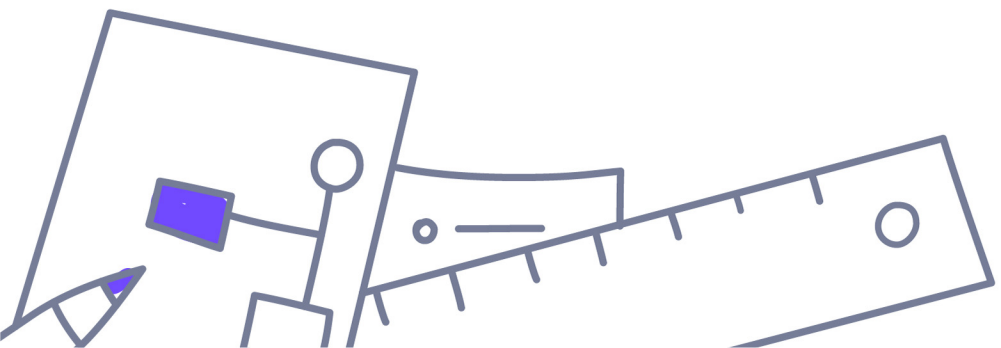
- Today we will be building and deploying a three-tier application together.
- The **Frontend** and **Server** are both deployed within the same **Azure Resource Group**.
- The server saves data within **Azure Cosmos DB**.
- The user interacts with the **Frontend**.

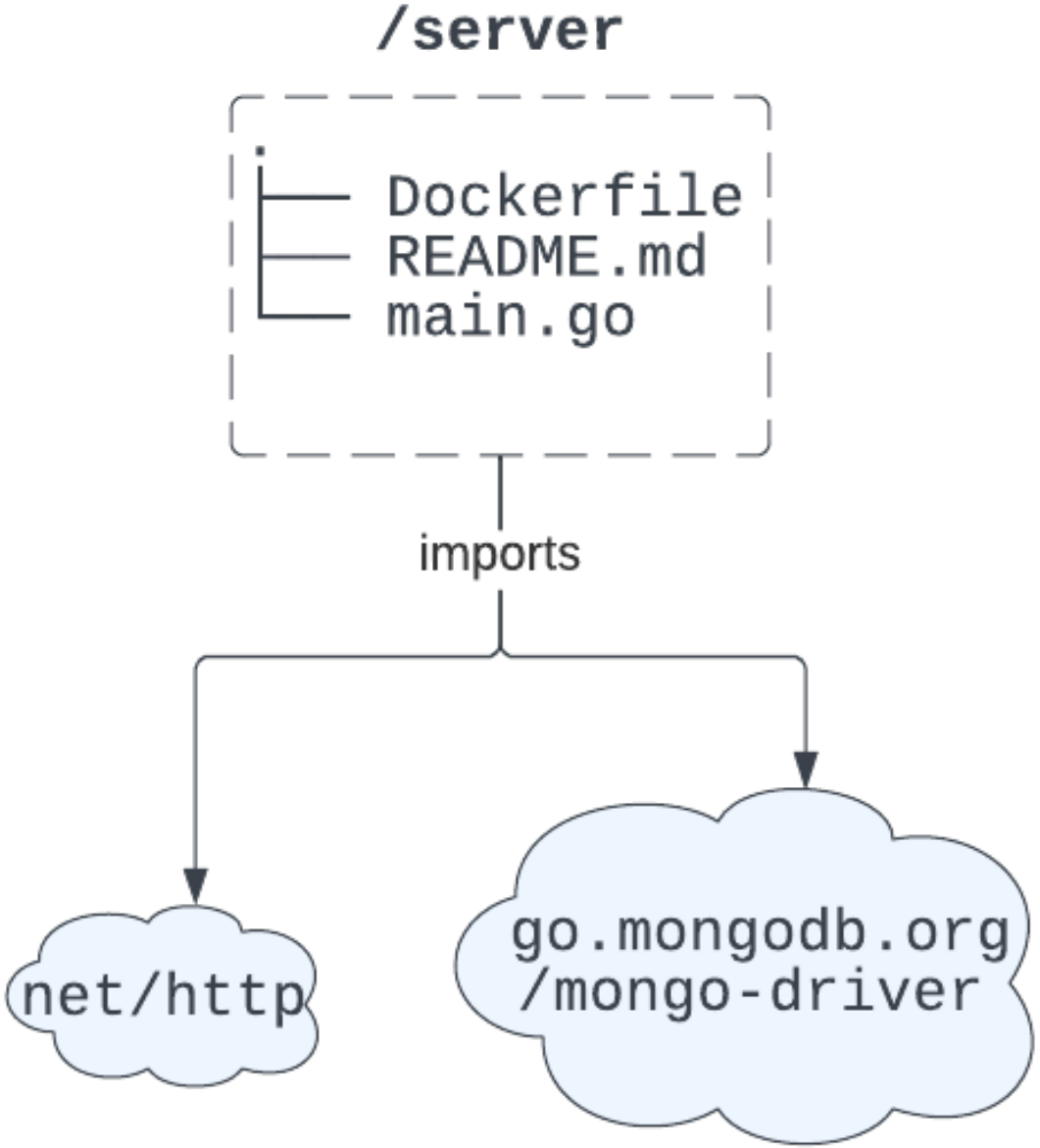




Everything is built with Go!

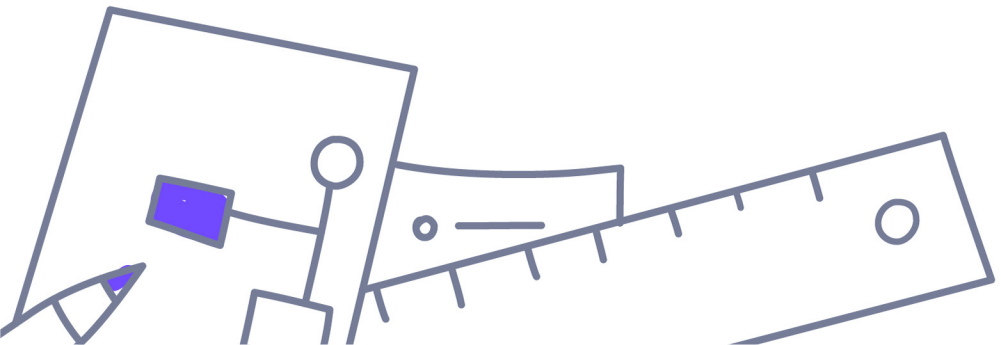
- The server exposes routes using **net/http**.
- **GET /persons** returns all elements from the database.
- **POST /persons** creates a new element and persists it to the database.
- The **MongoDB Go Driver** provides us an easy-to-use integration with CosmosDB.

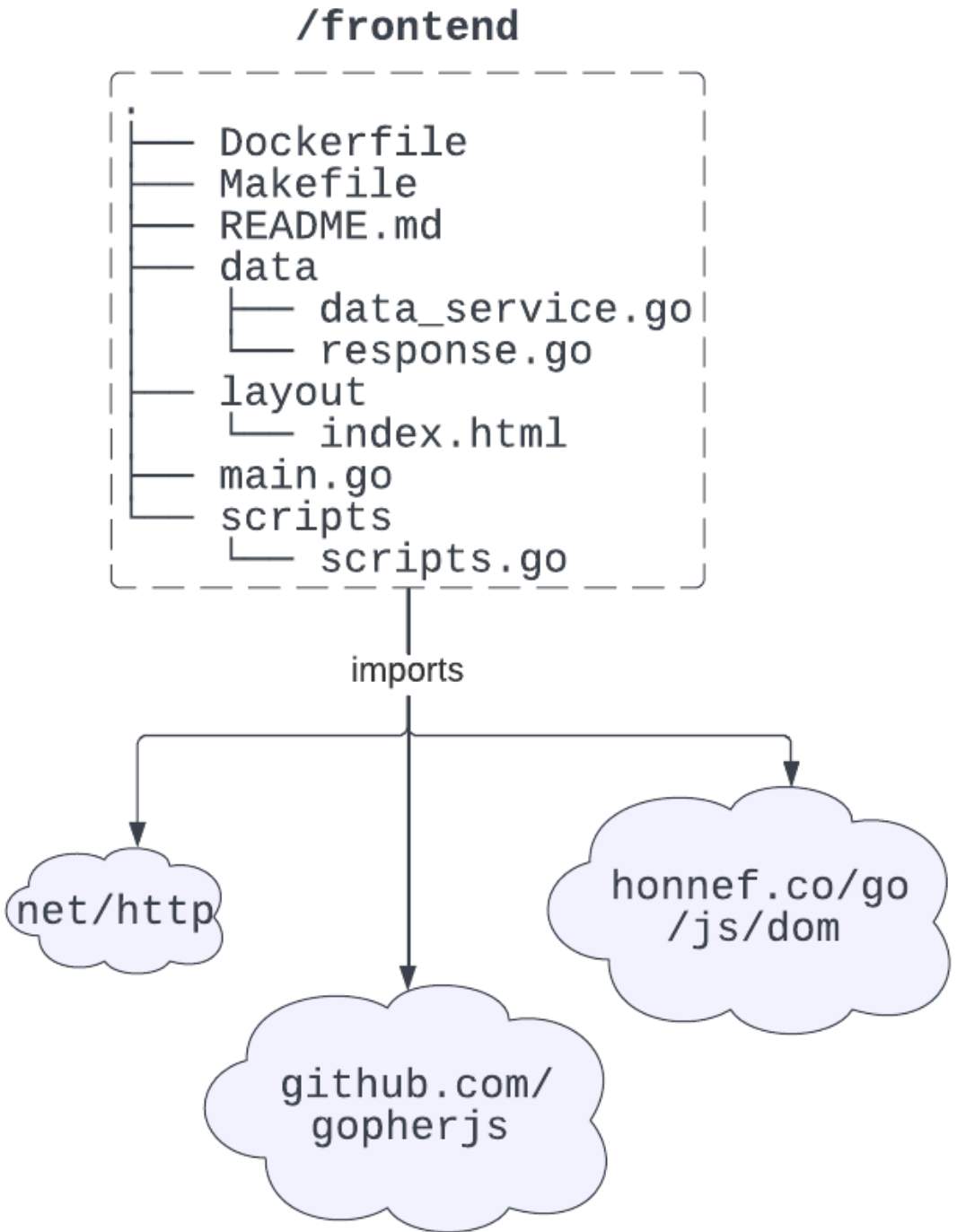




Everything is built with Go…even the Frontend!

- The **Frontend** server uses **net/http** to expose routes, exactly like the backend.
- It serves an HTML file on the **GET /** endpoint.
- The **js/dom** library allows us to model the HTML model tree in Go, making it easy to manipulate state.
- The **scripts.go** source file is used to generate JavaScript using **gopherjs**. This removes the need for us to write any JavaScript code.
- The **data_service.go** uses **net/http** to fetch data from the server application. JSON responses are unmarshalled and displayed.

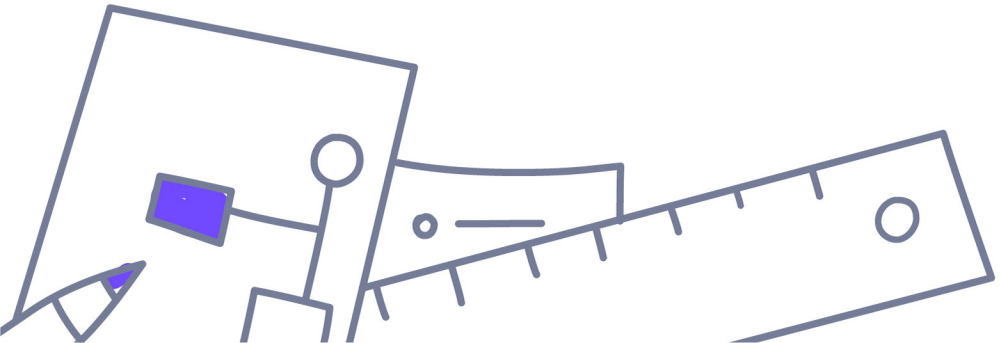




Implement the server

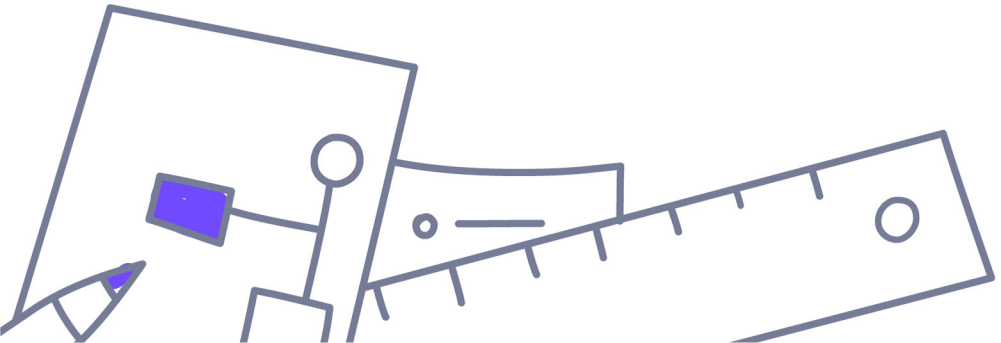
Cloud pipelines / Azure overview

- Using **GitHub Actions**, we can deploy our code seamlessly
- GitHub Actions integrates well **Azure**
- Only when we change a file in the **/server** directory will the workflow run
- Automate the following steps:
  - Build a Docker Image of the newly changed code
  - Tag the Docker Image
  - Login to Azure
  - Login to **Azure Container Registry**
  - Push the Docker Image to Azure Container Registry



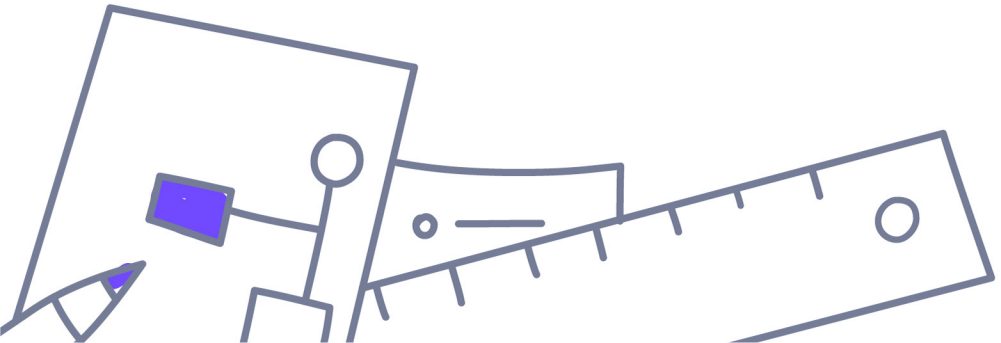
Cloud pipelines / Azure overview

- Using **GitHub Actions**, we can deploy our code seamlessly
- GitHub Actions integrates well **Azure**
- Only when we change a file in the **/server** directory will the workflow run
- Automate the following steps:
  - Build a Docker Image of the newly changed code
  - Tag the Docker Image
  - Login to Azure
  - Login to **Azure Container Registry**
  - Push the Docker Image to Azure Container Registry



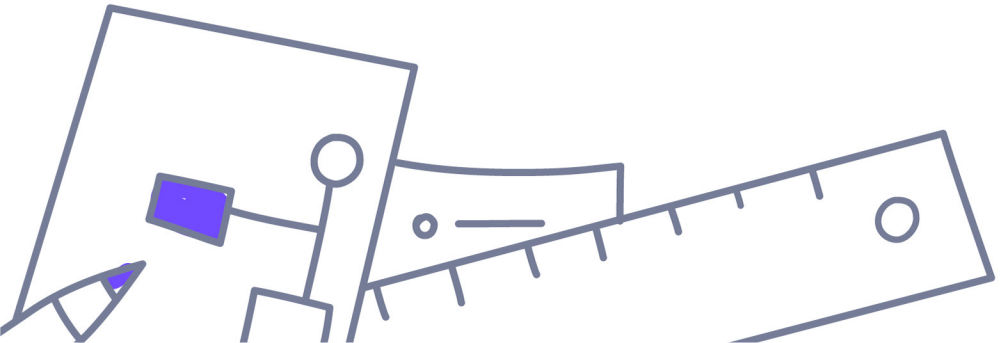
Cloud pipelines / Azure overview

- Using **GitHub Actions**, we can deploy our code seamlessly
- GitHub Actions integrates well **Azure**
- Only when we change a file in the **/server** directory will the workflow run
- Automate the following steps:
  - Build a Docker Image of the newly changed code
  - Tag the Docker Image
  - Login to Azure
  - Login to **Azure Container Registry**
  - Push the Docker Image to Azure Container Registry



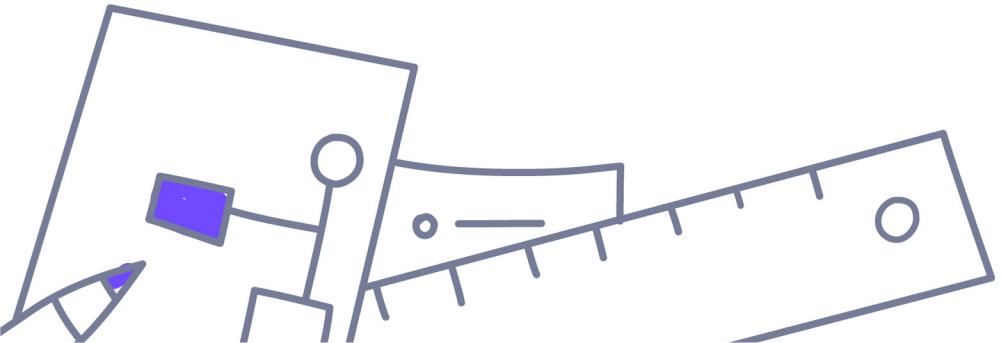
Cloud pipelines / Azure overview

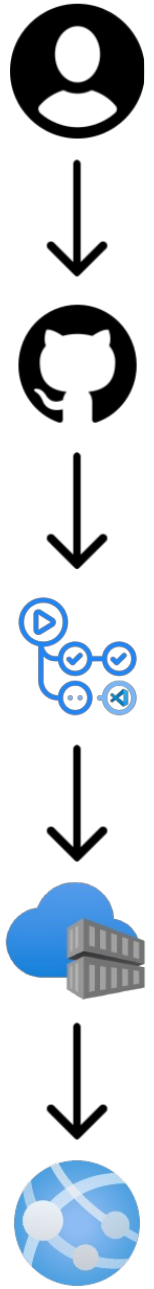
- Using **GitHub Actions**, we can deploy our code seamlessly
- GitHub Actions integrates well **Azure**
- Only when we change a file in the **/server** directory will the workflow run
- Automate the following steps:
  - Build a Docker Image of the newly changed code
  - Tag the Docker Image
  - Login to Azure
  - Login to **Azure Container Registry**
  - Push the Docker Image to Azure Container Registry



Cloud pipelines / Azure overview

- Using **GitHub Actions**, we can deploy our code seamlessly
- GitHub Actions integrates well **Azure**
- Only when we change a file in the **/server** directory will the workflow run
- Automate the following steps:
  - Build a Docker Image of the newly changed code
  - Tag the Docker Image
  - Login to Azure
  - Login to **Azure Container Registry**
  - Push the Docker Image to Azure Container Registry





Deploy our changes

Thank You!

Adelina Simion

Technology Evangelist @ Form3

@classic_addetz

Liam Hampton

Senior Regional Cloud Advocate @ Microsoft

@liamchampton

It looks like we have encountered a technical issue. We apologise for any inconvenience and will attempt to resume the event as soon as possible.

Thank you for your patience.

It looks like we have encountered a technical issue that prevents us from proceeding with this event.

We apologise for any inconvenience caused and will attempt to share the presentation post event.

Thank you for your patience.

FEEDBACK

Please don’t forget to leave your feedback via the survey posted in the chat. Thanks!