Appendix E

https://drive.google.com/file/d/1HP3g7Et2L_ImGYg-A49iTlwDQYSLtasP/view?usp=sharing

Figure 1.

- The video shows Frontend Service which is reached on equantifier.com:3000. The service takes you to the sign_up page, after sign_up you go to the email verification page, and lastly to the home page.
- There's a Users service which handles user registration
- There's a Verification service which handles sending verification links
- RabbitMQ is used for async communication
- MongoDB stores verification links
- Postgres has users and tracing data
- Redis has running service IDs

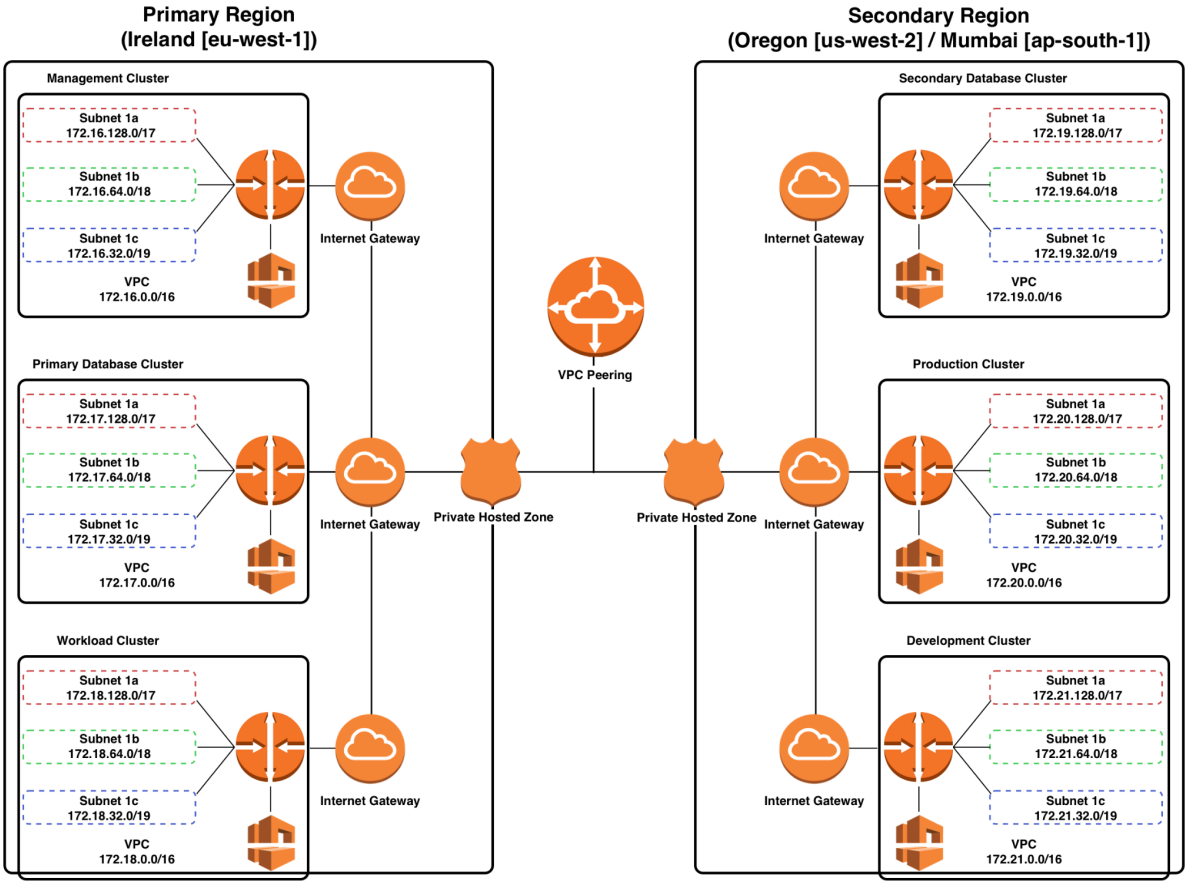

Figure 2.

Multi-Regional AWS Architecture

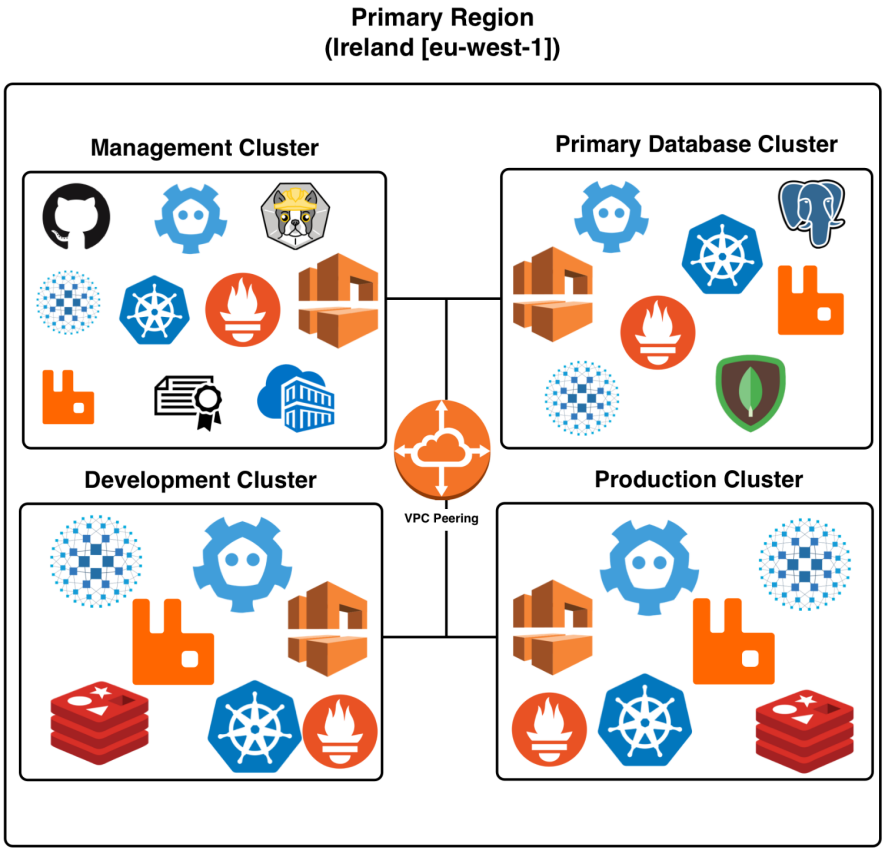

Figure 3.

Software Stack on different VPCs in the primary region

```go
package resources

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// +kubebuilder:subresource:status
// +kubebuilder:resource:scope="Cluster",shortName=etcd
// +kubebuilder:printcolumn:name="Cluster",type=string,JSONPath=`.spec.cluster`
// +kubebuilder:printcolumn:name="Members",type=integer,JSONPath=`.status.members`
type ETCDCluster struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   ETCDClusterSpec   `json:"spec"`
    Status ETCDClusterStatus `json:"status,omitempty"`
}

type ETCDClusterSpec struct {
    // +kubebuilder:validation:MinLength:2
    // +kubebuilder:validation:MaxLength:4
    Cluster string `json:"cluster"`
    // +kubebuilder:validation:Optional
    // +kubebuilder:validation:Minimum:1
    Members int `json:"members"`
}

type ETCDClusterStatus struct {
    //Condition [string]string
    Members      int                      `json:"members"`
    Certificates ETCDClusterCertificates `json:"certificates"`
}

type ETCDClusterCertificates struct {
    CA     Certificate `json:"ca"`
    Client Certificate `json:"client"`
}
```

Figure 4.
Golang ETCD to create and monitor ETCD as a Kubernetes resource

```
@Injectable
export class CreateInstanceSaga extends EC2Service {

    svc = `CreateInstanceSaga`
    private defaults = {…
    }

    constructor(){
        super()
    }

    get handler(){
        const cluster = new SagaHandler(this.cluster,'cluster').addInput(readCluster.output).asynk()
        const securityGroups = new SagaHandler(this.securityGroups,'securityGroups').persist('securityGroupIds')
        const category = new SagaHandler(this.category,'category').persist('count')
        const subnet = new SagaHandler(this.subnet,'subnet').addCompensation(this.subnetComp).persist('subnetId')
        const runInstance = new SagaHandler(this.runInstance,'runInstance').addCompensation(this.runInstanceComp).persist('instanceId')
        const awaitInstance = new SagaHandler(this.awaitInstance,'awaitInstance')
        const saveInstance = new SagaHandler(this.saveInstance,'saveInstance').addCompensation(this.saveInstanceComp)
        const onComplete = new SagaHandler(this.onComplete,'onComplete')
        return new SagaBuilder<Data>(logger,createInstance)
                        .push(cluster,securityGroups,category,subnet,runInstance,awaitInstance,saveInstance,onComplete)
                        .handler
    }

    private onComplete = async (data:Data,asynk:Asynk,req:AMQPInReq)=> {…
    }

    private cluster = async (data:Data,asynk:Asynk,req:AMQPInReq,replyTo:OriginalRequest)=> {…
    }

    private securityGroups = async (data:Data):Promise<Data['securityGroups']>=>{…
    }

    private category = async (data:Data):Promise<Data['category']>=> {…
    }

    private subnet = async (data:Data,asynk:Asynk,req:AMQPInReq):Promise<Data['subnet']>=>{…
    }

    private subnetComp = async (data:Data)=>{…
    }

    private runInstance = async (data:Data):Promise<Data['runInstance']>=>{…
    }
```

Figure 5.
Saga for creating an ec2 instance.

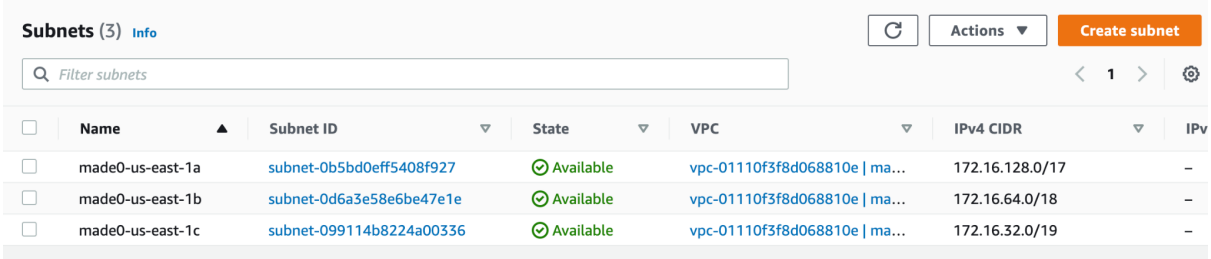

Figure 6.
Subnets on AWS