Generating a TLS certificate

Let's use Cert Manager to generate the TLS certificate that our API gateway will use to enable HTTPS and TLS termination.

In Play.Infra

1. Add a cert-manager directory

2. Add cluster-issuer.yaml:

Issuers, and ClusterIssuers, are Kubernetes resources that represent certificate authorities (CAs) that are able to generate signed certificates by honoring certificate signing requests.

Let's Encrypt is a nonprofit Certificate Authority providing TLS certificates

ACME = Automated Certificate Management Environment

The ACME Issuer type represents a single account registered with the ACME Certificate Authority server

This ACME protocol which supports various challenge mechanisms for verifying ownership of the domain.

The HTTP-01 challenge verifies ownership of the domain by sending a request for a specific file on that domain. cert-manager accomplishes this by sending a request to a temporary pod in our Kubernetes cluster.

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    # The ACME server URL
    server: https://acme-v02.api.letsencrypt.org/directory
    # Email address used for ACME registration
    email: julioc@dotnetmicroservices.com
    # Secret used to store the ACME account private key, which is used to identify you with the ACME
server.
    privateKeySecretRef:
      name: letsencrypt-prod
    # Enable the HTTP-01 challenge provider
    solvers:
    - http01:
        ingress:
          class:  nginx
```

3. Update README:

Creating the cluster issuer
```powershell
kubectl apply -f .\cert-manager\cluster-issuer.yaml -n $namespace
```

4. Run the command

5. Verify cluster-issuer is ready:

   kubectl get clusterissuer -n emissary

“Now, since we will be using the http01 ACME challenge, we know that cert-manager will spin up a temporal pod in our cluster with the file that the challenge http request will be looking for.

But, for that http request to reach our temporal pod from the outside world, we need to have a Kubernetes service that can select the pod that will receive the request and, since we use the emissary-ingress API Gateway, we also need a mapping that will route requests from the API Gateway to this service.

6. Add acme-challenge.yaml (cert-manager directory):

```
apiVersion: v1
kind: Service
metadata:
  name: acme-challenge-service
spec:
  ports:
  - port: 80
    targetPort: 8089
  selector:
    acme.cert-manager.io/http01-solver: "true"

---
apiVersion: getambassador.io/v3alpha1
kind: Mapping
metadata:
  name: acme-challenge-mapping
spec:
  hostname: playeconomy.eastus.cloudapp.azure.com
  prefix: /.well-known/acme-challenge/
  rewrite: ""
  service: acme-challenge-service
```

7. Update README:

    ## Creating the cluster issuer
    ```powershell
    kubectl apply -f .\cert-manager\cluster-issuer.yaml -n $namespace
    **kubectl apply -f .\emissary-ingress\acme-challenge.yaml -n $namespace**
    ```

8. Run the command

9. Add tls-certificate.yaml (emissary-ingress directory):

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: playeconomy-tls-cert
spec:
  secretName: playeconomy-tls
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
  dnsNames:
  - playeconomy.eastus.cloudapp.azure.com
```

10. Update README:

    ## Creating the tls certificate
    ```powershell
    kubectl apply -f .\cert-manager\tls-certificate.yaml -n $namespace
    ```

11. Run the command

12. Verify certificate and secret are ready:

    kubectl get certificate -n $namespace
    kubectl describe certificate playeconomy-cert -n $namespace
    kubectl get secret -n $namespace

13. Commit and push

In the next lesson you will enable HTTPS and TLS termination in your Emissary-ingress API gateway.