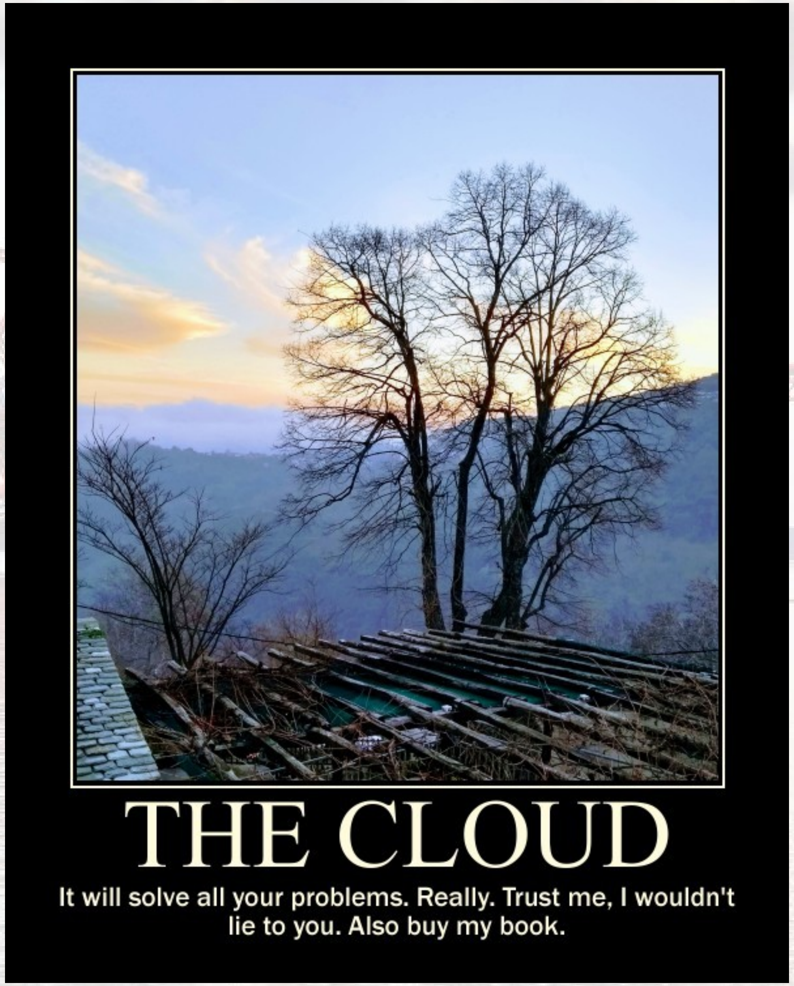
Deploying PostgreSQL on Kubernetes

Jimmy Angelakos
Platform Architect
SolarWinds MSP

FOSDEM
03/02/2019

Motivation

- Service Oriented Architecture (SOA), including Micro– , exemplified perfectly by Kubernetes
- Kubernetes is here to stay
- Fewer phonecalls at 4 am?
- Play around at home for free
- Or get commercial support
- Cloud Compute, Storage → Commodity
- (Industrial-strength) Postgres is hard
- You want Postgres → Commodity to your users
- By no means an exhaustive list of solutions or in-depth analysis but an attempt to demystify


THE CLOUD
It will solve all your problems. Really. Trust me, I wouldn't lie to you. Also buy my book.

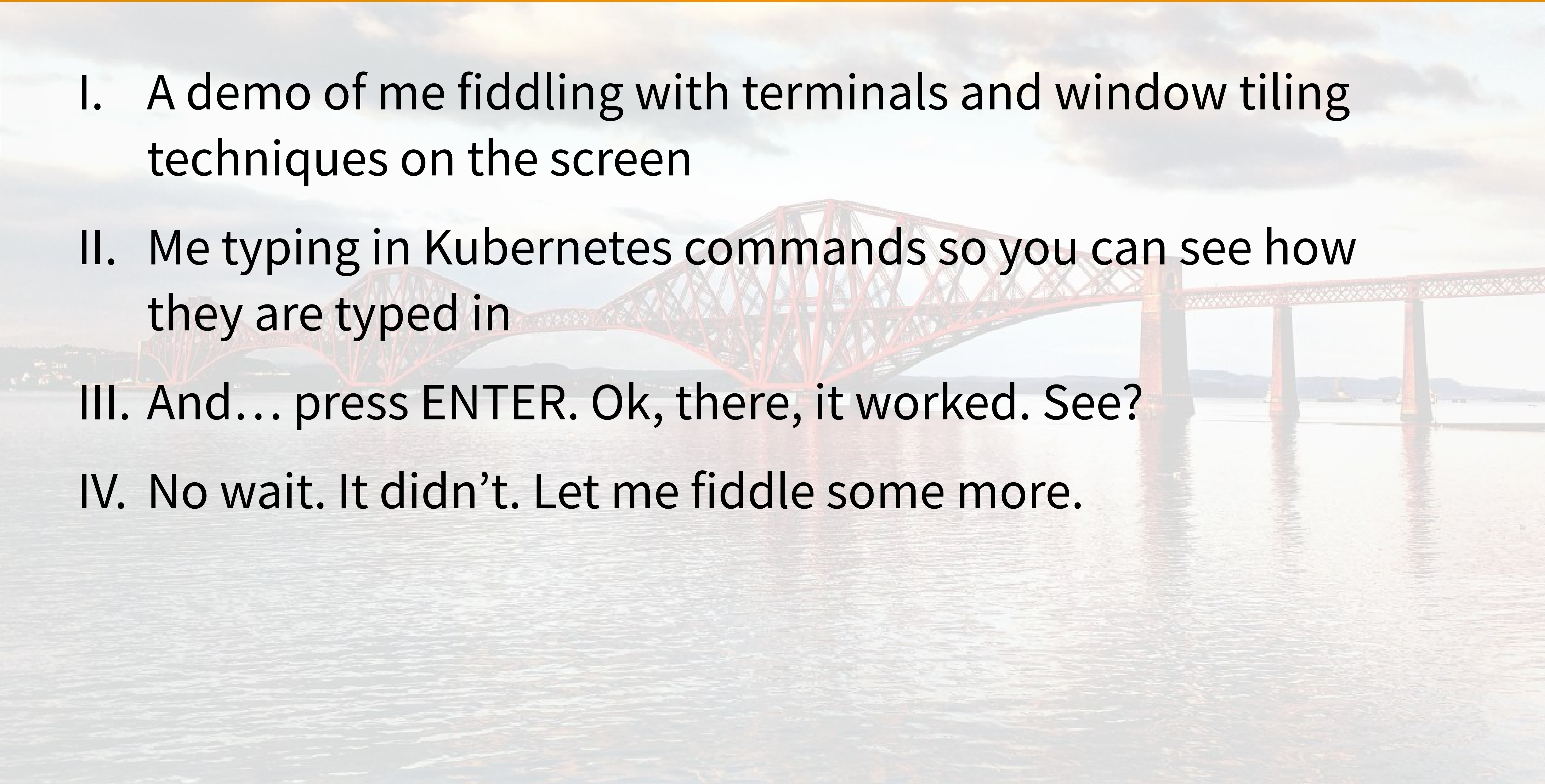
What this is not

I. A demo of me fiddling with terminals and window tiling techniques on the screen

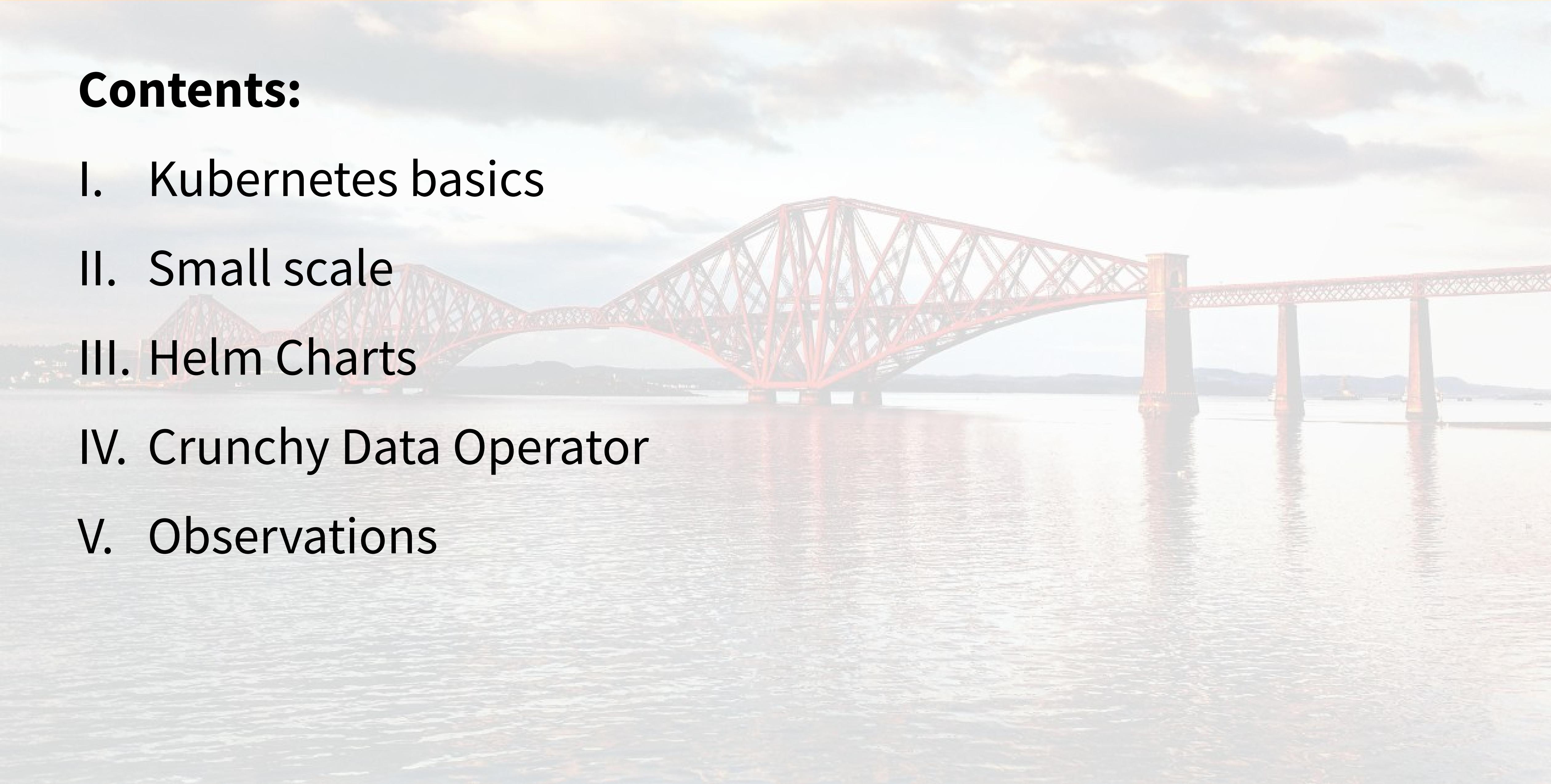
II. Me typing in Kubernetes commands so you can see how they are typed in

III. And… press ENTER. Ok, there, it worked. See?

IV. No wait. It didn’t. Let me fiddle some more.



What this is



Contents:

I. Kubernetes basics

II. Small scale

III. Helm Charts

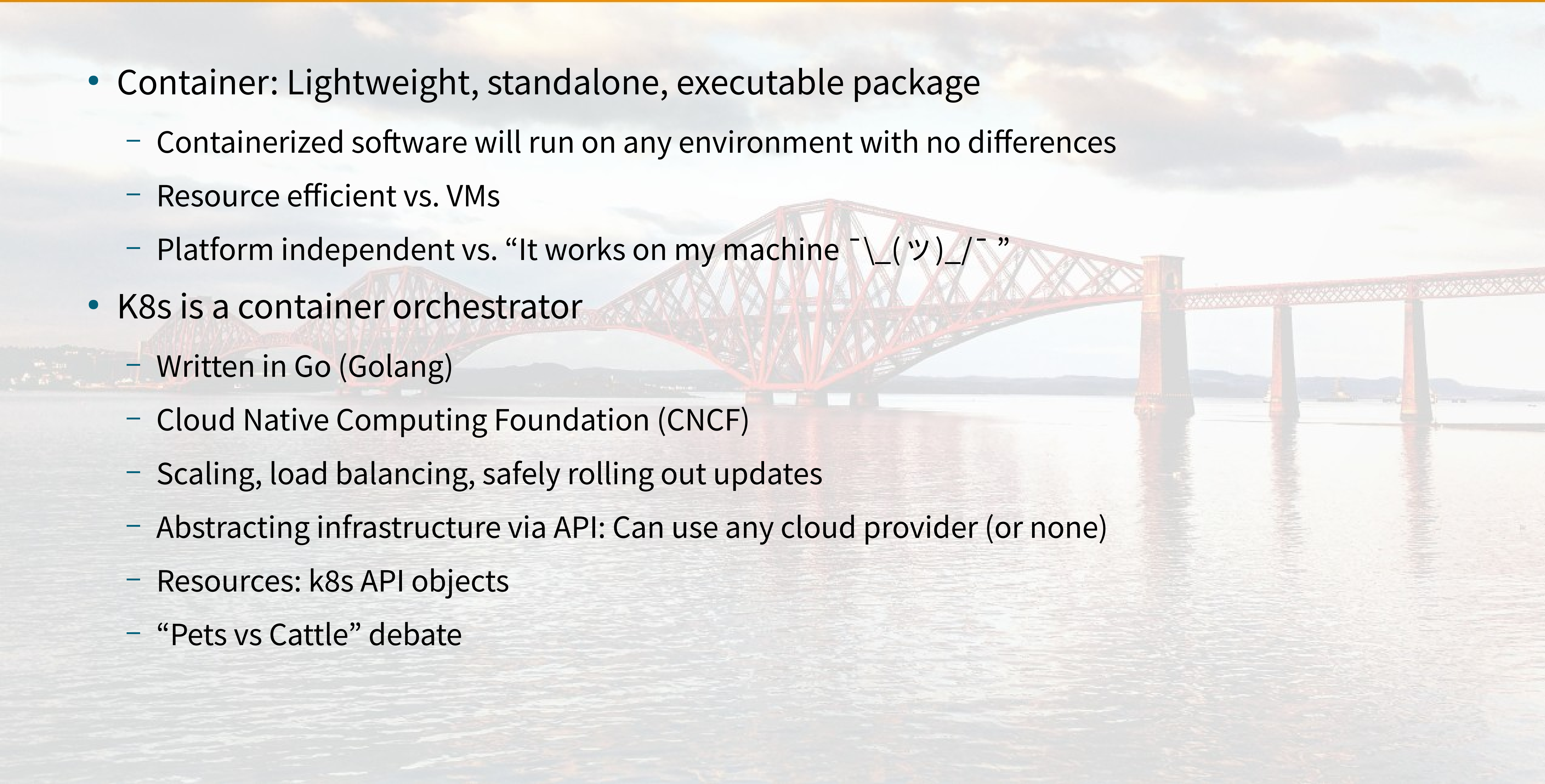
IV. Crunchy Data Operator

V. Observations

I.

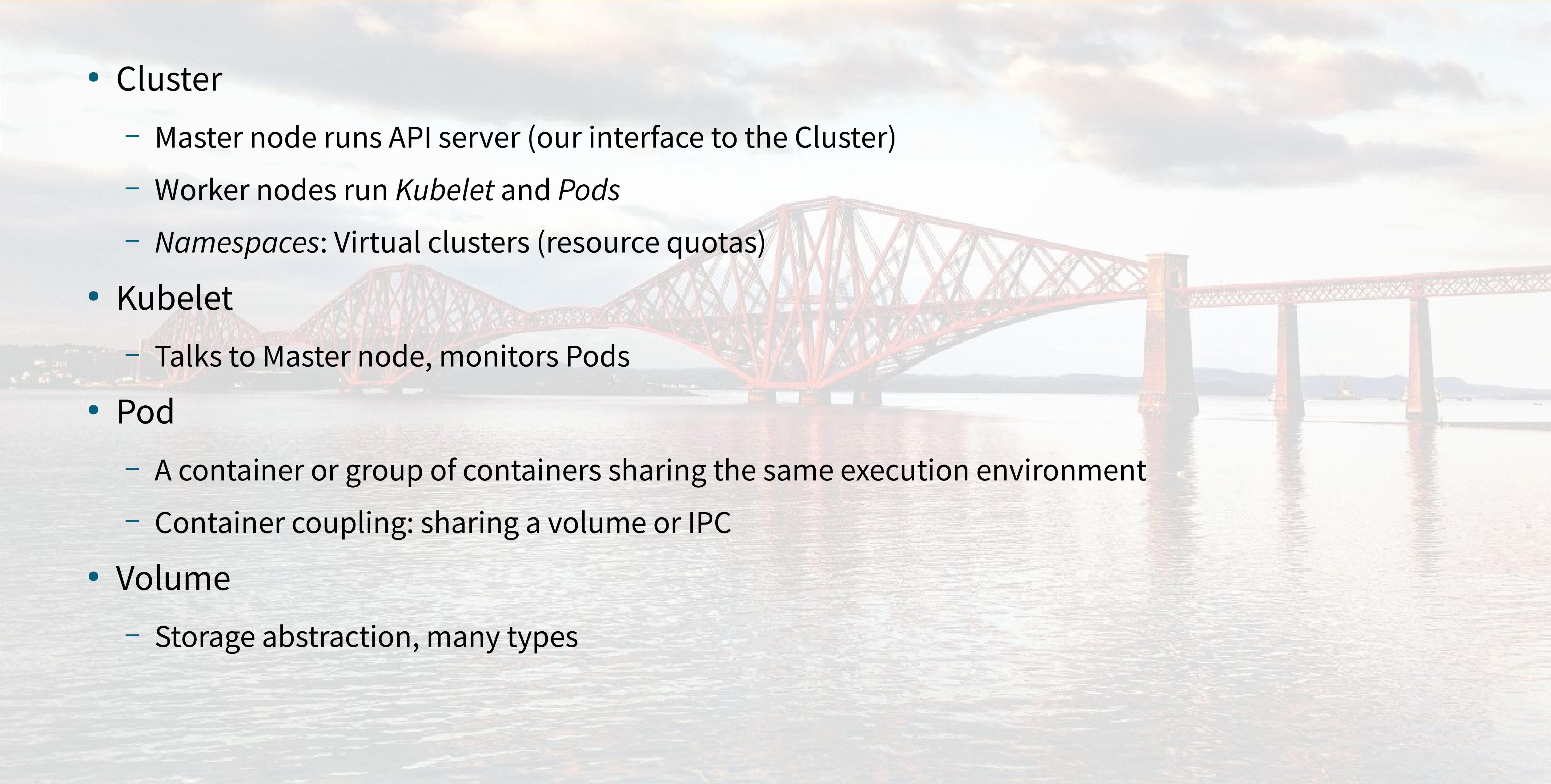
Kubernetes (k8s) basics

K8s basics – 1: K8s & Containers



- Container: Lightweight, standalone, executable package
  - Containerized software will run on any environment with no differences
  - Resource efficient vs. VMs
  - Platform independent vs. “It works on my machine ¯_(ツ)_/¯ ”
- K8s is a container orchestrator
  - Written in Go (Golang)
  - Cloud Native Computing Foundation (CNCF)
  - Scaling, load balancing, safely rolling out updates
  - Abstracting infrastructure via API: Can use any cloud provider (or none)
  - Resources: k8s API objects
  - “Pets vs Cattle” debate

K8s basics – 2: Terms



- Cluster
  - Master node runs API server (our interface to the Cluster)
  - Worker nodes run *Kubelet* and *Pods*
  - *Namespaces*: Virtual clusters (resource quotas)
- Kubelet
  - Talks to Master node, monitors Pods
- Pod
  - A container or group of containers sharing the same execution environment
  - Container coupling: sharing a volume or IPC
- Volume
  - Storage abstraction, many types

K8s basics – 3: Moar terms

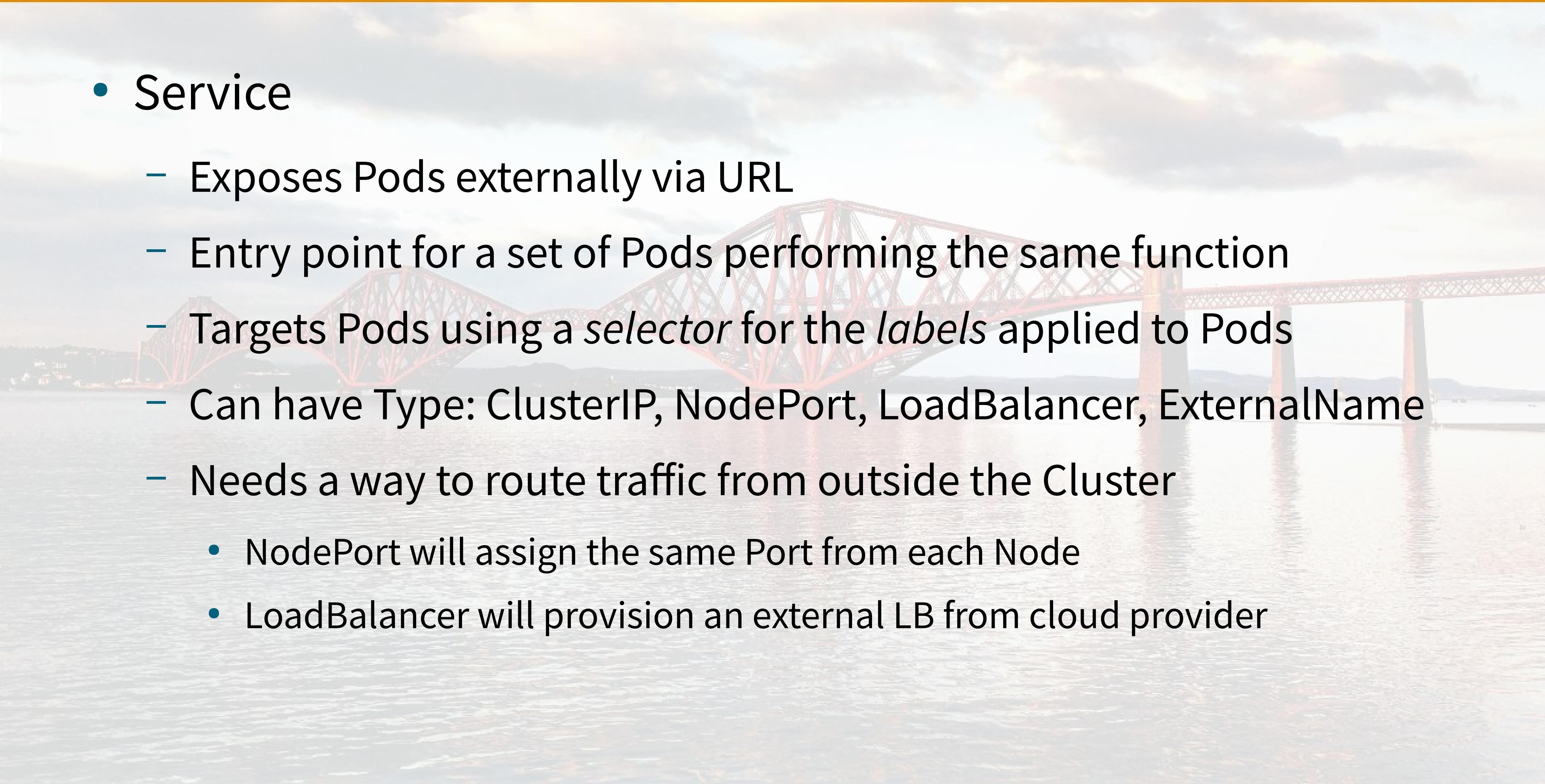
- Minikube
  - Single-node k8s cluster in a VM – install VirtualBox and you're good to go.
- Prometheus
  - Monitoring solution for k8s (also by CNCF, so described as "best fit"…)
- Custom Resource Definitions
  - Write them to extend k8s API at will
- Operator pattern
  - Custom domain-specific controllers that work with CRDs
  - Configure & manage stateful applications for you
  - No need for out-of-band automation

K8s basics – 4: YAML files

- Definitions
  - **YAML!**
  - `kind` of resource e.g. `Pod`
  - `metadata` e.g. `name`, `labels`
  - `spec` i.e. the desired state for the resource
- Kubectl
  - CLI tool for interacting with Cluster

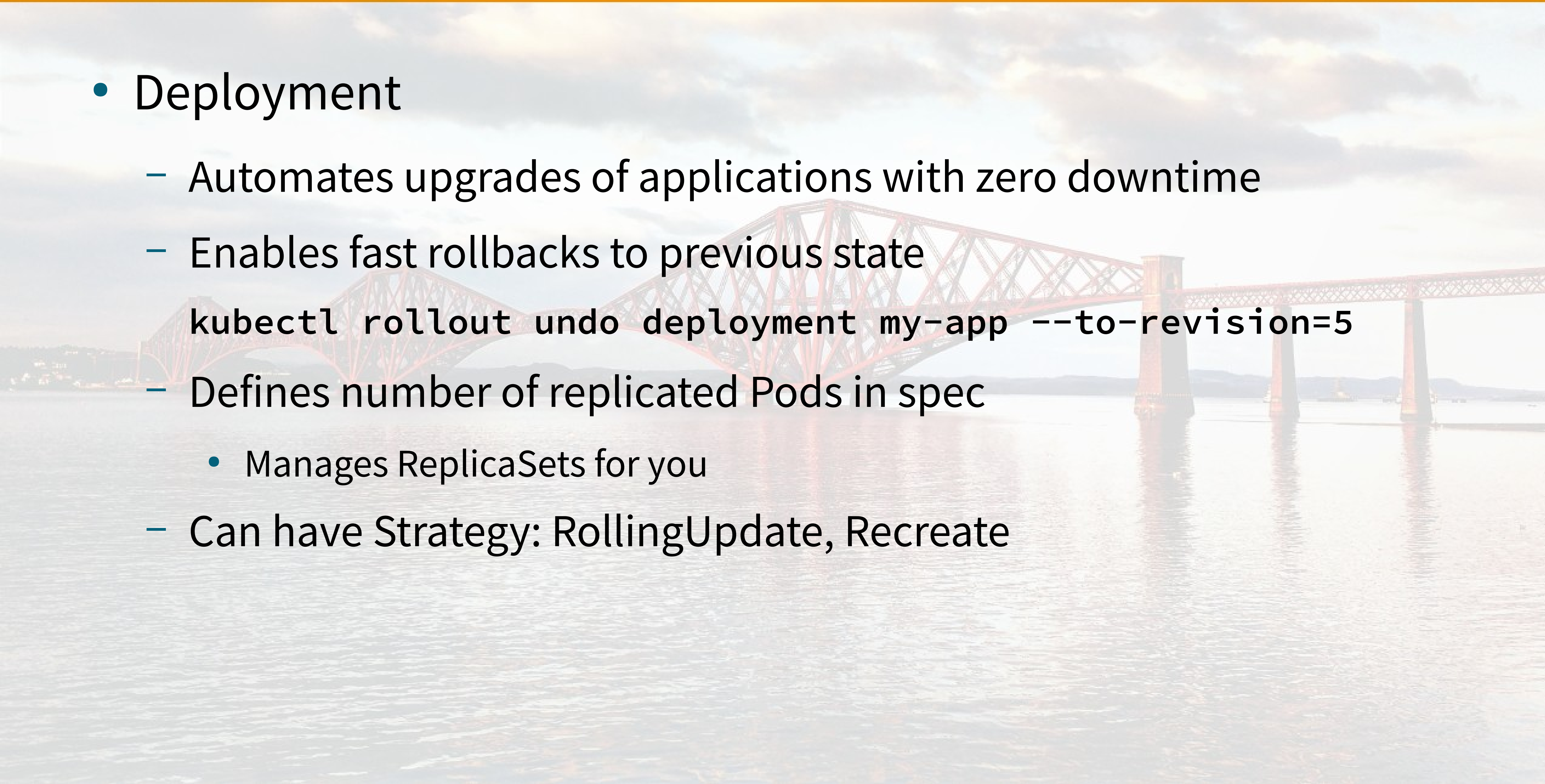
```
kubectl create -f my-pod.yaml
kubectl get pods
```

K8s basics – 5: Services



- Service
  - Exposes Pods externally via URL
  - Entry point for a set of Pods performing the same function
  - Targets Pods using a *selector* for the *labels* applied to Pods
  - Can have Type: ClusterIP, NodePort, LoadBalancer, ExternalName
  - Needs a way to route traffic from outside the Cluster
    - NodePort will assign the same Port from each Node
    - LoadBalancer will provision an external LB from cloud provider

K8s basics – 6: Deployments



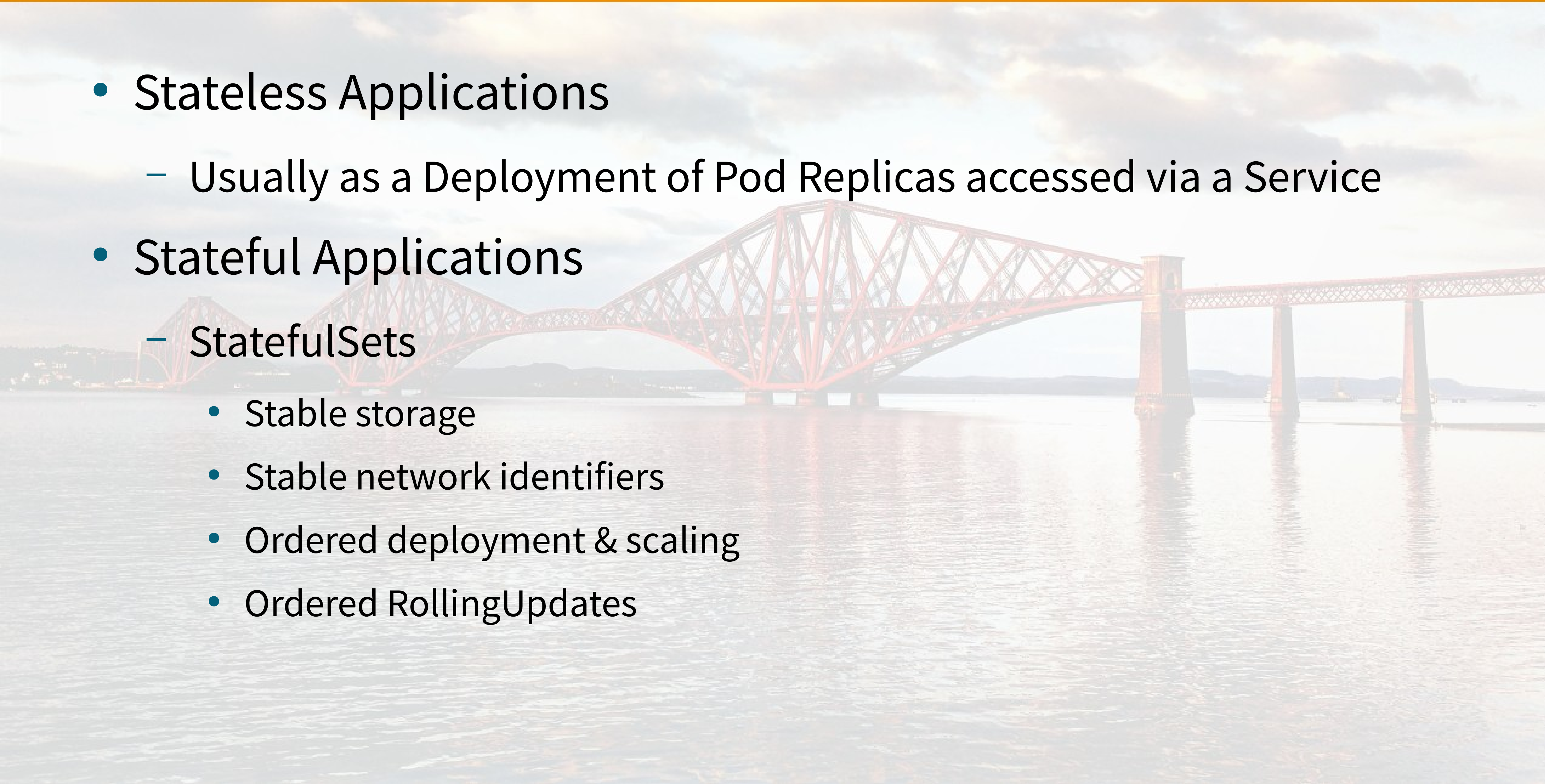
- Deployment
  - Automates upgrades of applications with zero downtime
  - Enables fast rollbacks to previous state

```
kubectl rollout undo deployment my-app --to-revision=5
```

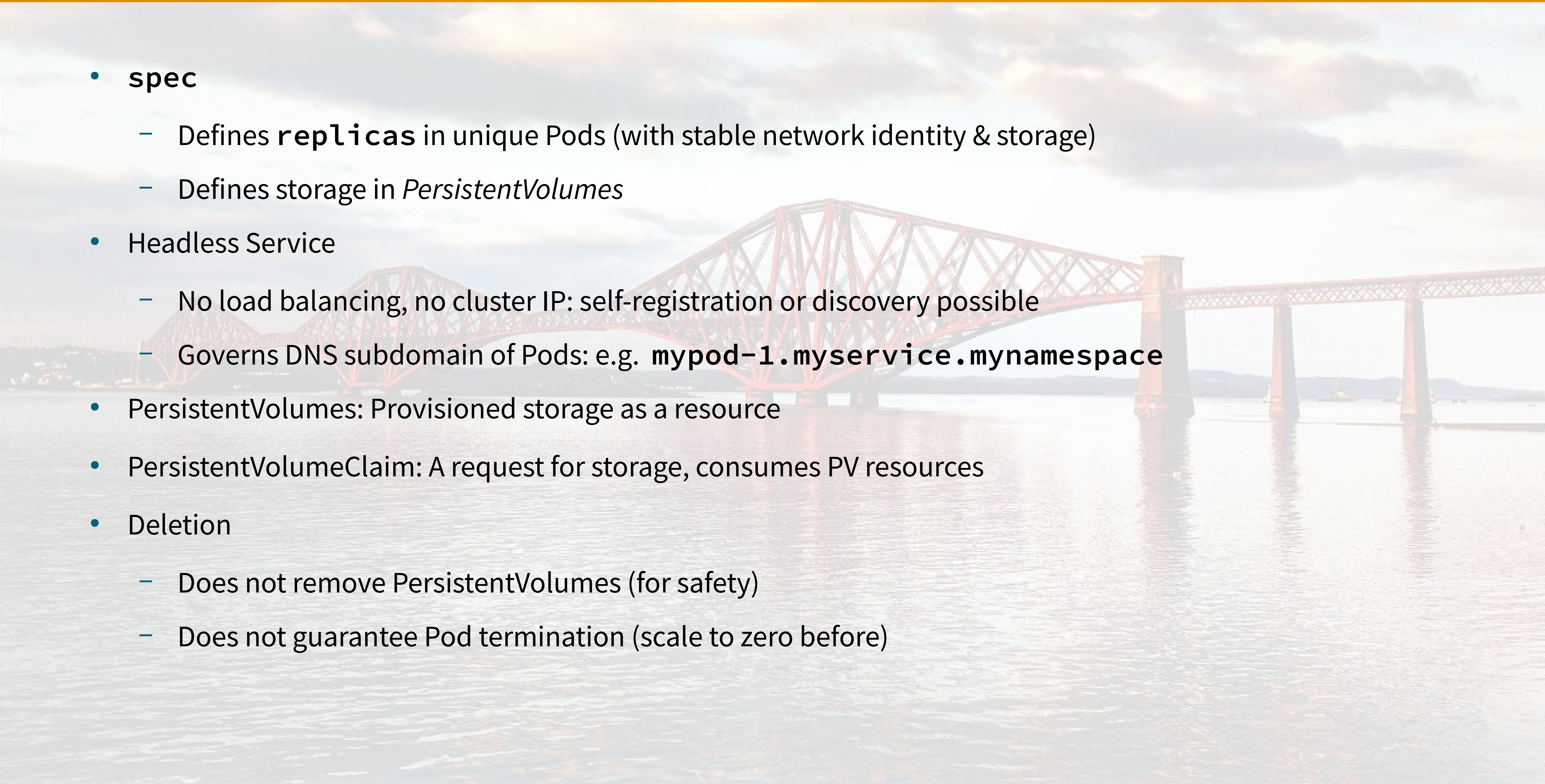
  - Defines number of replicated Pods in spec
    - Manages ReplicaSets for you
  - Can have Strategy: RollingUpdate, Recreate

K8s basics – 7: State

- Stateless Applications
  - Usually as a Deployment of Pod Replicas accessed via a Service
- Stateful Applications
  - StatefulSets
    - Stable storage
    - Stable network identifiers
    - Ordered deployment & scaling
    - Ordered RollingUpdates



K8s basics – 8: StatefulSets

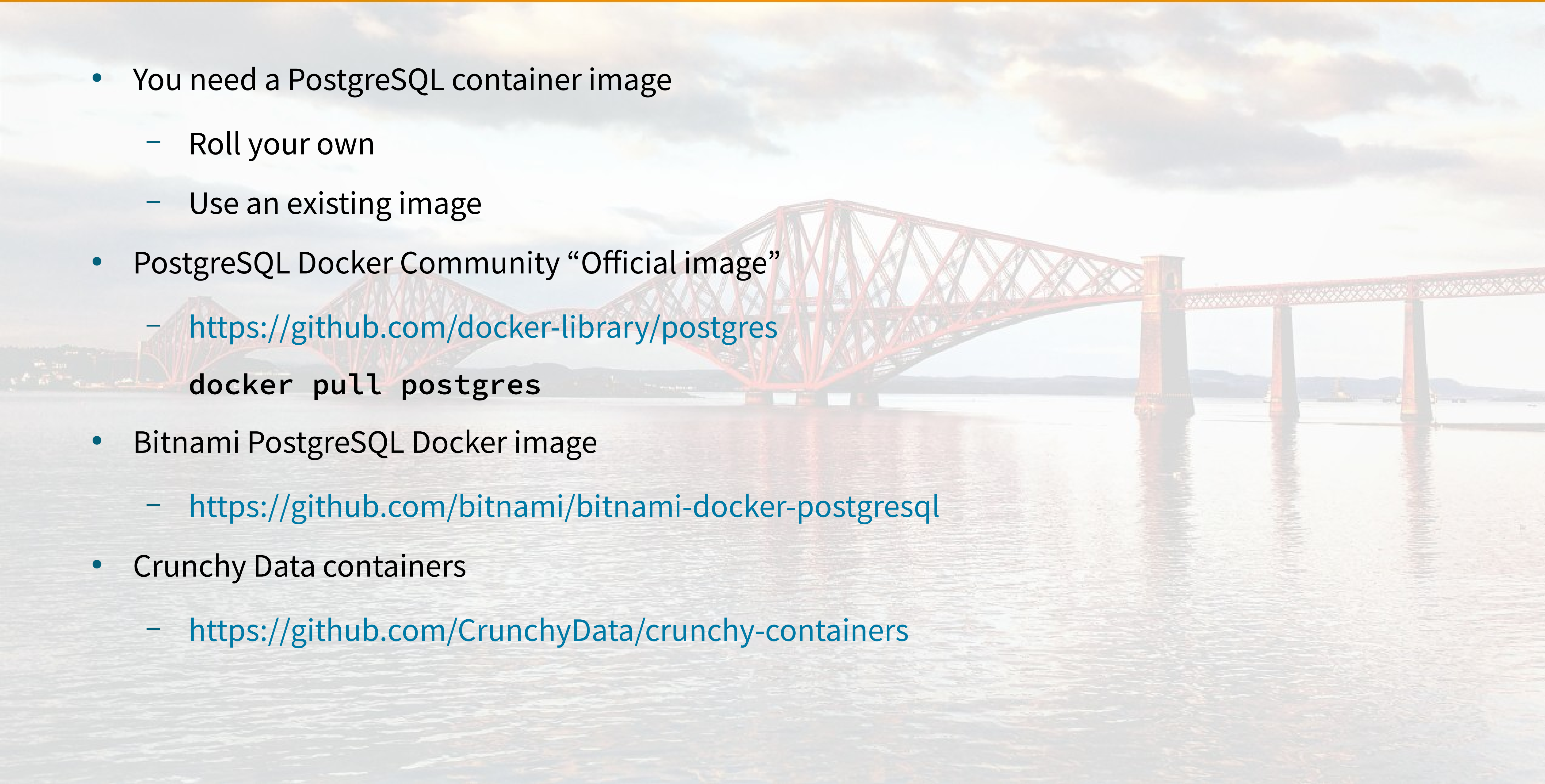


- `spec`
  - Defines **`replicas`** in unique Pods (with stable network identity & storage)
  - Defines storage in *PersistentVolumes*
- Headless Service
  - No load balancing, no cluster IP: self-registration or discovery possible
  - Governs DNS subdomain of Pods: e.g. **`mypod-1.myservice.mynamespace`**
- PersistentVolumes: Provisioned storage as a resource
- PersistentVolumeClaim: A request for storage, consumes PV resources
- Deletion
  - Does not remove PersistentVolumes (for safety)
  - Does not guarantee Pod termination (scale to zero before)

II.

Small scale

Small scale – 1: The image

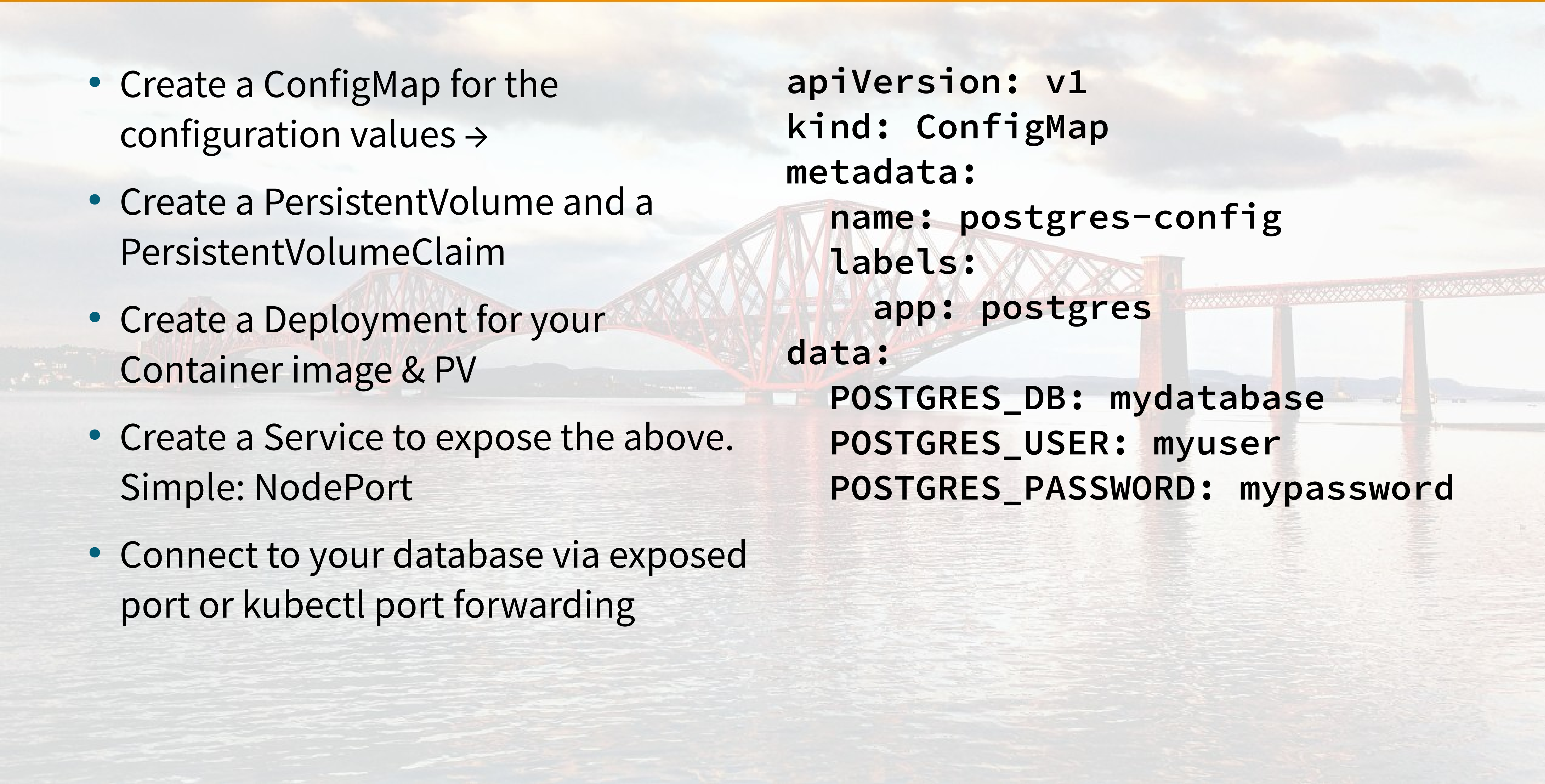


- You need a PostgreSQL container image
  - Roll your own
  - Use an existing image
- PostgreSQL Docker Community “Official image”
  - https://github.com/docker-library/postgres

```
docker pull postgres
```

- Bitnami PostgreSQL Docker image
  - https://github.com/bitnami/bitnami-docker-postgresql
- Crunchy Data containers
  - https://github.com/CrunchyData/crunchy-containers

Small scale – 2: Deployment



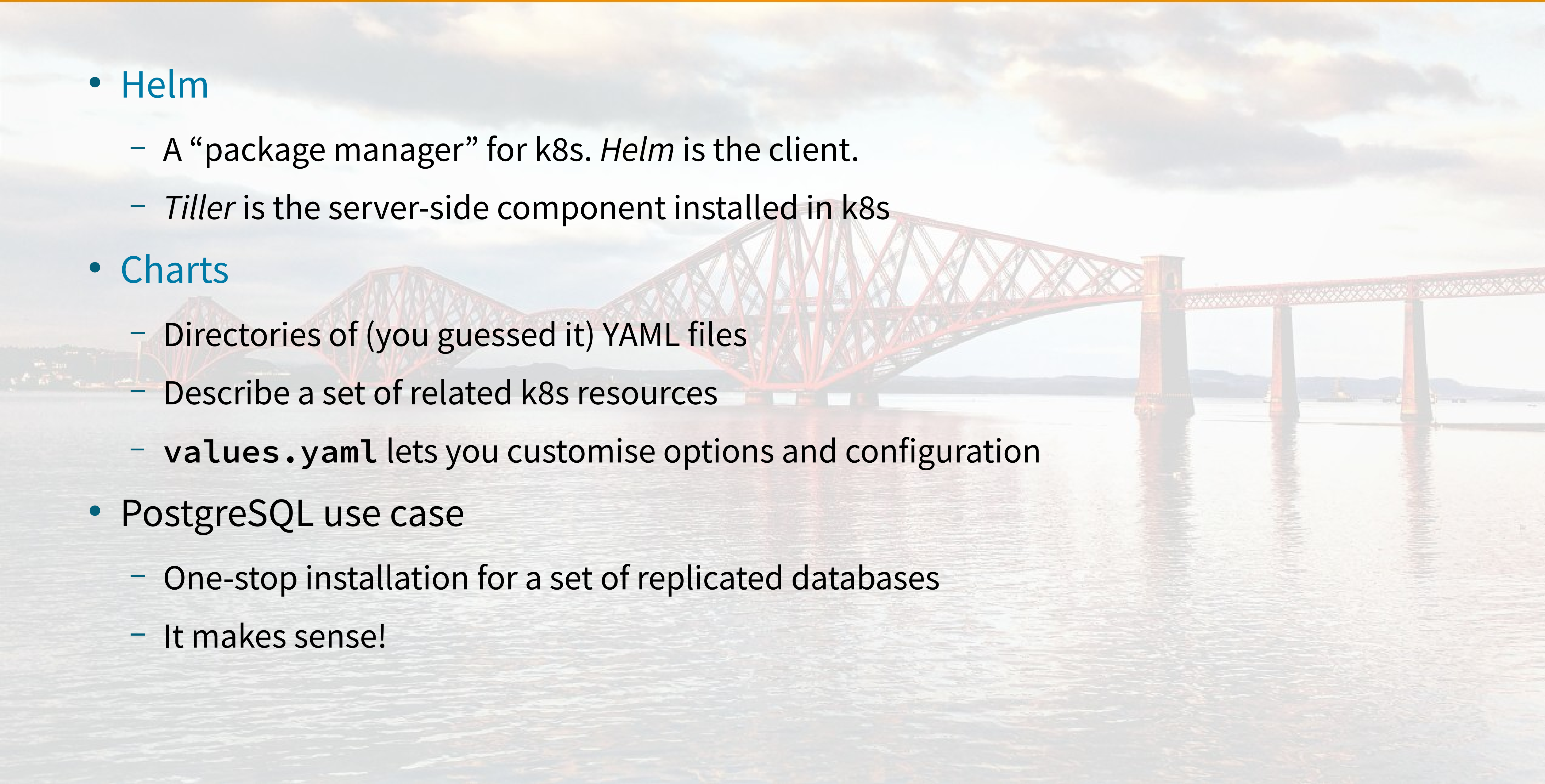
- Create a ConfigMap for the configuration values →
- Create a PersistentVolume and a PersistentVolumeClaim
- Create a Deployment for your Container image & PV
- Create a Service to expose the above. Simple: NodePort
- Connect to your database via exposed port or kubectl port forwarding

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config
  labels:
    app: postgres
data:
  POSTGRES_DB: mydatabase
  POSTGRES_USER: myuser
  POSTGRES_PASSWORD: mypassword
```

III.

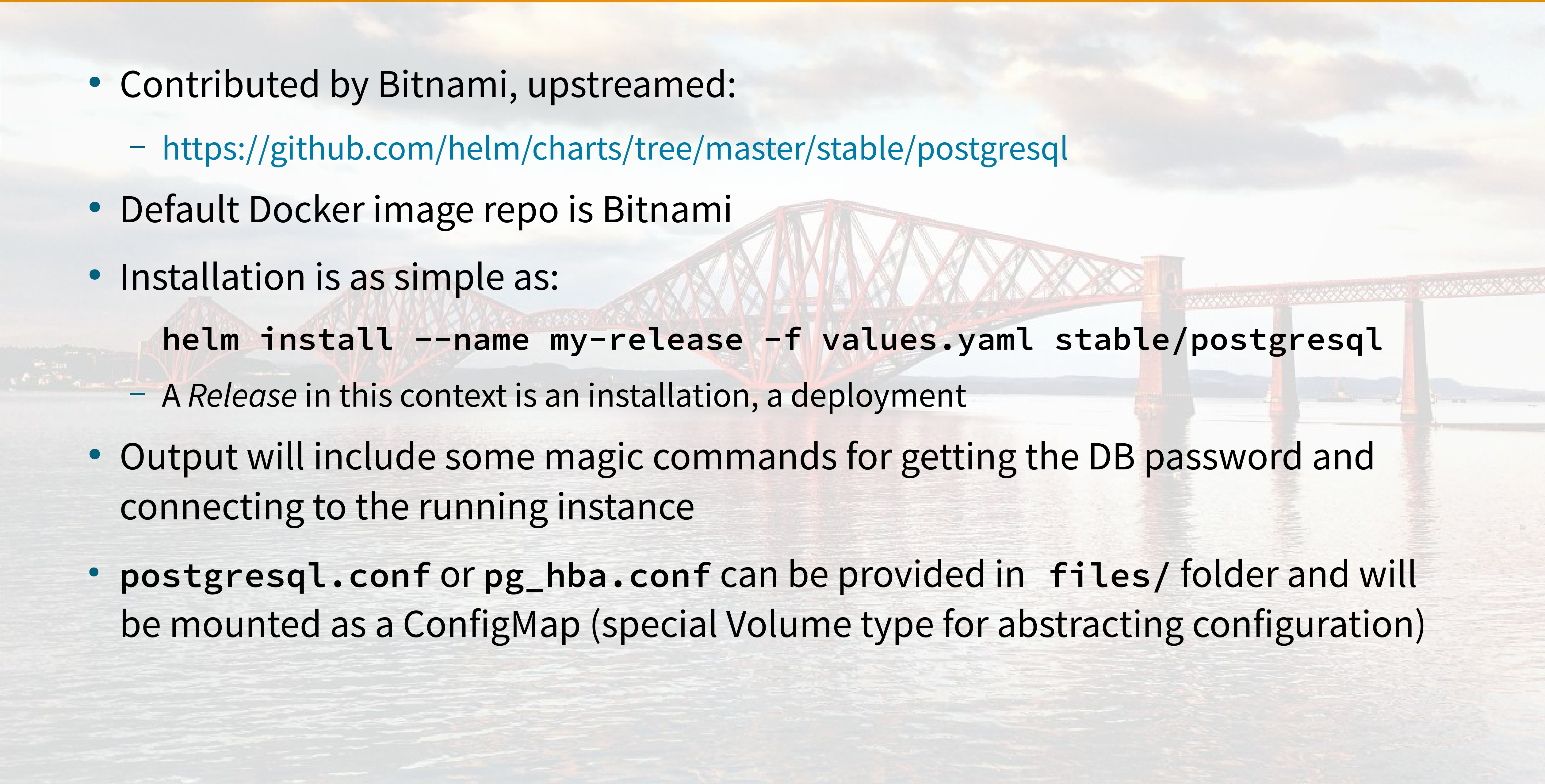
Helm Charts

Helm Charts – 1: Introduction



- Helm
  - A “package manager” for k8s. *Helm* is the client.
  - *Tiller* is the server-side component installed in k8s
- Charts
  - Directories of (you guessed it) YAML files
  - Describe a set of related k8s resources
  - **`values.yaml`** lets you customise options and configuration
- PostgreSQL use case
  - One-stop installation for a set of replicated databases
  - It makes sense!

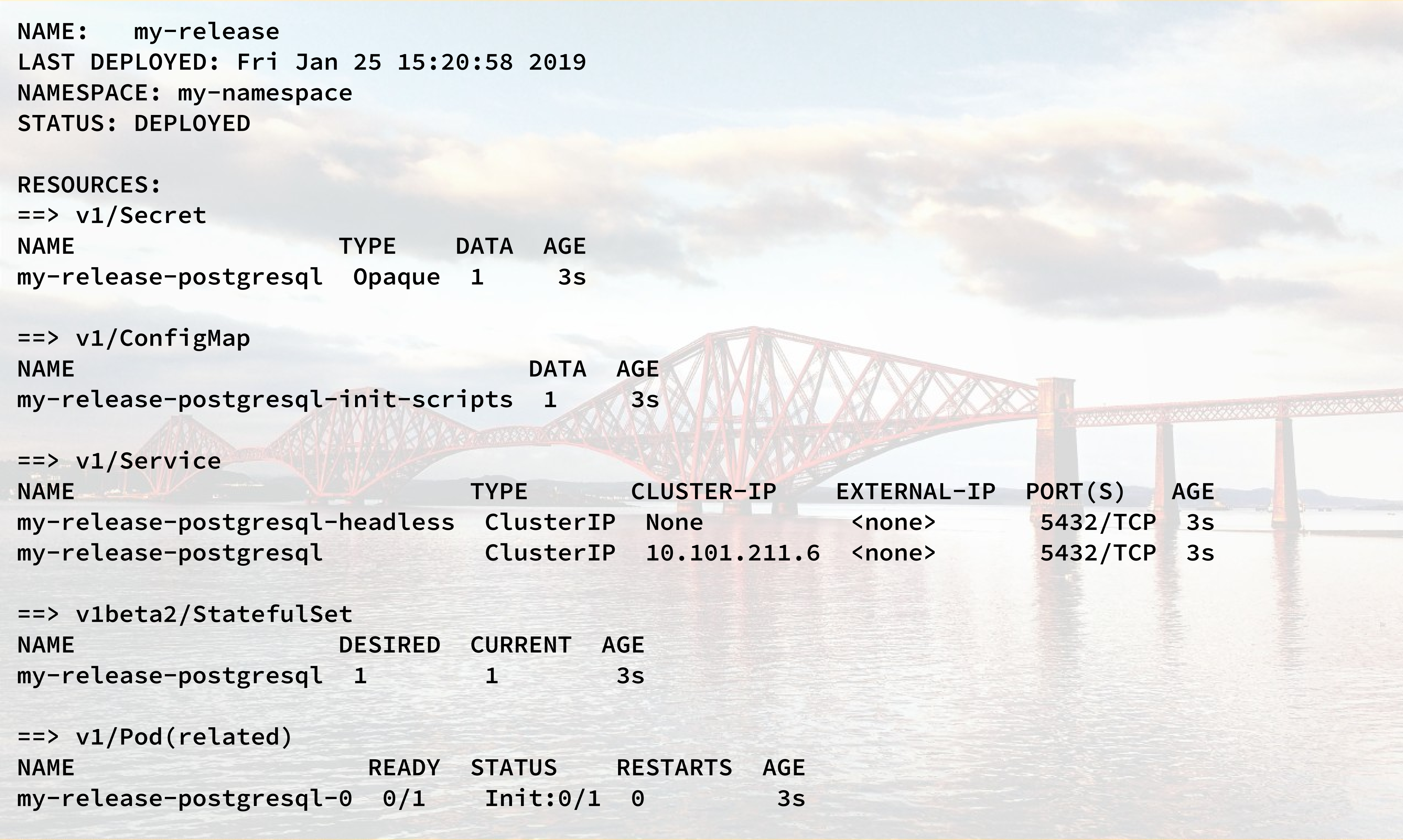
Helm Charts – 2: PostgreSQL Chart



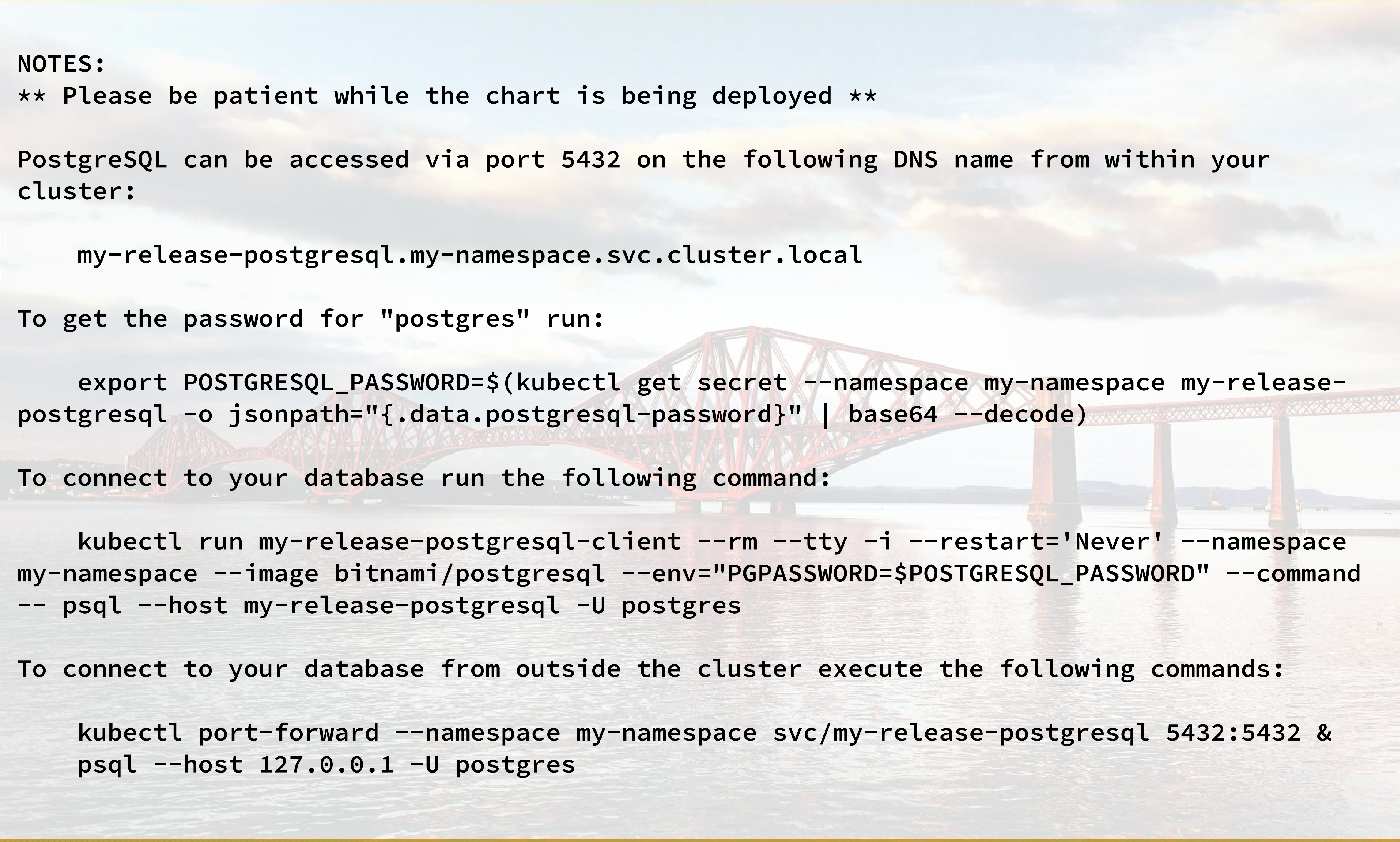
- Contributed by Bitnami, upstreamed:
  - https://github.com/helm/charts/tree/master/stable/postgresql
- Default Docker image repo is Bitnami
- Installation is as simple as:

```
helm install --name my-release -f values.yaml stable/postgresql
```

  - A *Release* in this context is an installation, a deployment
- Output will include some magic commands for getting the DB password and connecting to the running instance
- **`postgresql.conf`** or **`pg_hba.conf`** can be provided in **`files/`** folder and will be mounted as a ConfigMap (special Volume type for abstracting configuration)



```
NAME:   my-release
LAST DEPLOYED: Fri Jan 25 15:20:58 2019
NAMESPACE: my-namespace
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                  TYPE    DATA  AGE
my-release-postgresql  Opaque  1     3s

==> v1/ConfigMap
NAME                               DATA  AGE
my-release-postgresql-init-scripts  1     3s

==> v1/Service
NAME                           TYPE       CLUSTER-IP    EXTERNAL-IP  PORT(S)   AGE
my-release-postgresql-headless  ClusterIP  None          <none>       5432/TCP  3s
my-release-postgresql           ClusterIP  10.101.211.6  <none>       5432/TCP  3s

==> v1beta2/StatefulSet
NAME                  DESIRED  CURRENT  AGE
my-release-postgresql  1        1        3s

==> v1/Pod(related)
NAME                    READY  STATUS    RESTARTS  AGE
my-release-postgresql-0  0/1    Init:0/1  0         3s
```

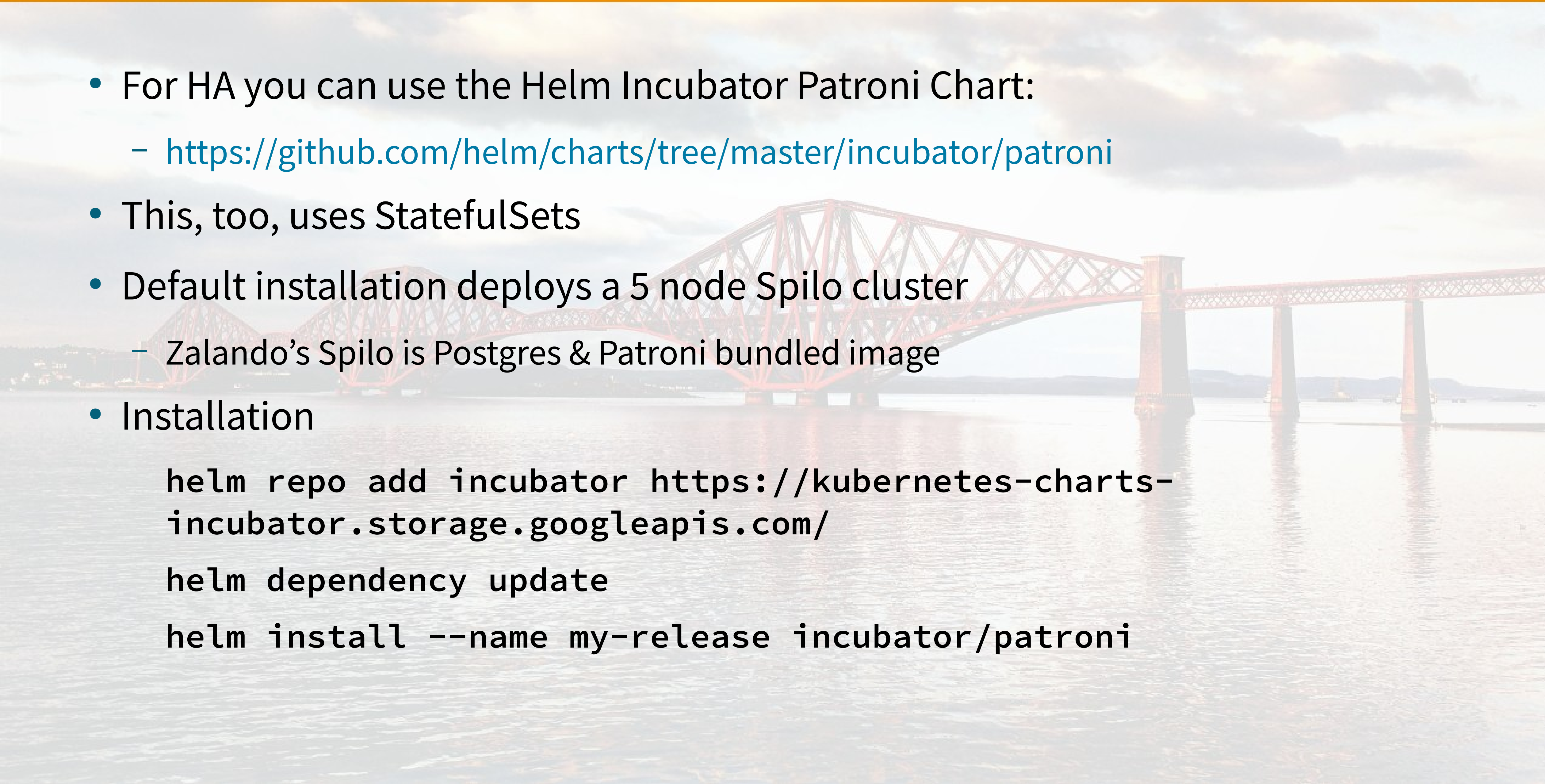


```
NOTES:
** Please be patient while the chart is being deployed **

PostgreSQL can be accessed via port 5432 on the following DNS name from within your cluster:

    my-release-postgresql.my-namespace.svc.cluster.local

To get the password for "postgres" run:

    export POSTGRESQL_PASSWORD=$(kubectl get secret --namespace my-namespace my-release-postgresql -o jsonpath="{.data.postgresql-password}" | base64 --decode)

To connect to your database run the following command:

    kubectl run my-release-postgresql-client --rm --tty -i --restart='Never' --namespace my-namespace --image bitnami/postgresql --env="PGPASSWORD=$POSTGRESQL_PASSWORD" --command -- psql --host my-release-postgresql -U postgres

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace my-namespace svc/my-release-postgresql 5432:5432 &
    psql --host 127.0.0.1 -U postgres
```

Helm Charts – 3: Internals

- Defaults create:
  - A StatefulSet with 1 Replica (1 Pod) running Postgres from the Docker image
  - A Headless Service and a Service
  - A PersistentVolumeClaim from the configured storage provisioner
- Can be configured to:
  - Load custom Postgres initialisation scripts as ConfigMaps from `files/`
  - Start a metrics exporter to Prometheus:
    - https://github.com/wrouesnel/postgres_exporter
    - Export e.g. `pg_stat_activity`, `pg_stat_replication` or custom metrics queries

Helm Charts – 4: Patroni Chart



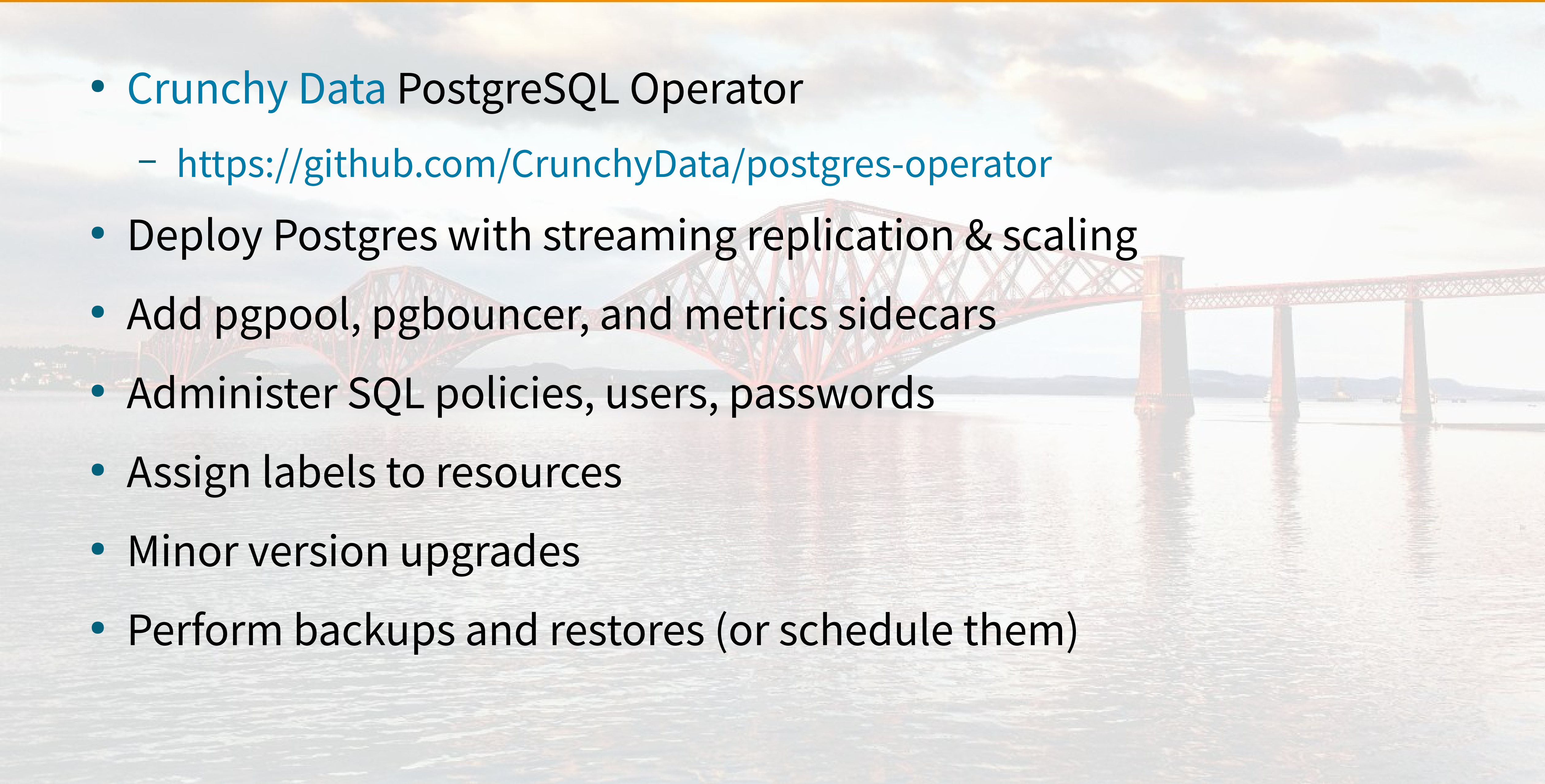
- For HA you can use the Helm Incubator Patroni Chart:
  - https://github.com/helm/charts/tree/master/incubator/patroni
- This, too, uses StatefulSets
- Default installation deploys a 5 node Spilo cluster
  - Zalando’s Spilo is Postgres & Patroni bundled image
- Installation

```
helm repo add incubator https://kubernetes-charts-
incubator.storage.googleapis.com/
helm dependency update
helm install --name my-release incubator/patroni
```

IV.

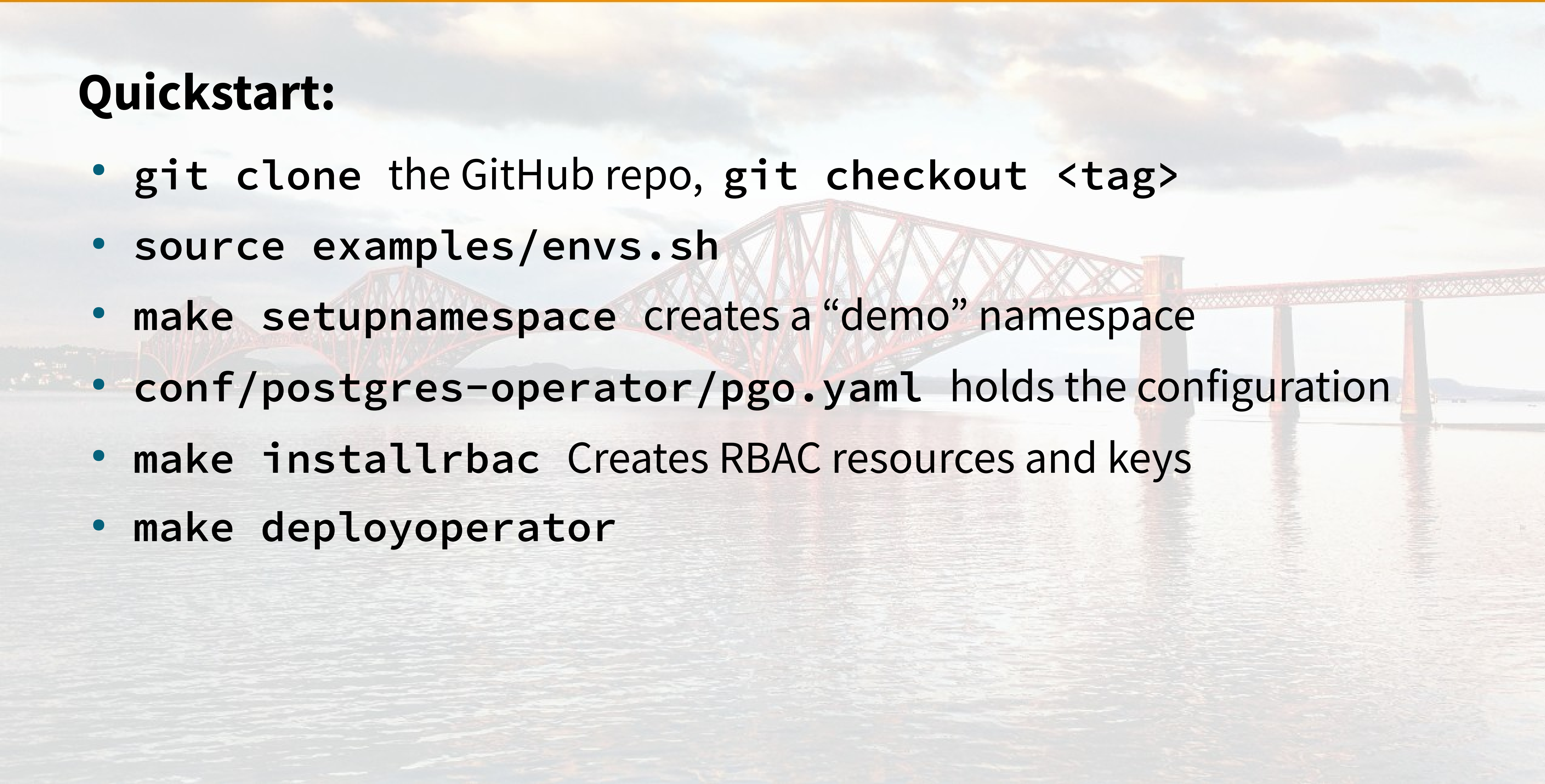
Crunchy Operator

Crunchy Operator – 1



- Crunchy Data PostgreSQL Operator
  - https://github.com/CrunchyData/postgres-operator
- Deploy Postgres with streaming replication & scaling
- Add pgpool, pgbouncer, and metrics sidecars
- Administer SQL policies, users, passwords
- Assign labels to resources
- Minor version upgrades
- Perform backups and restores (or schedule them)

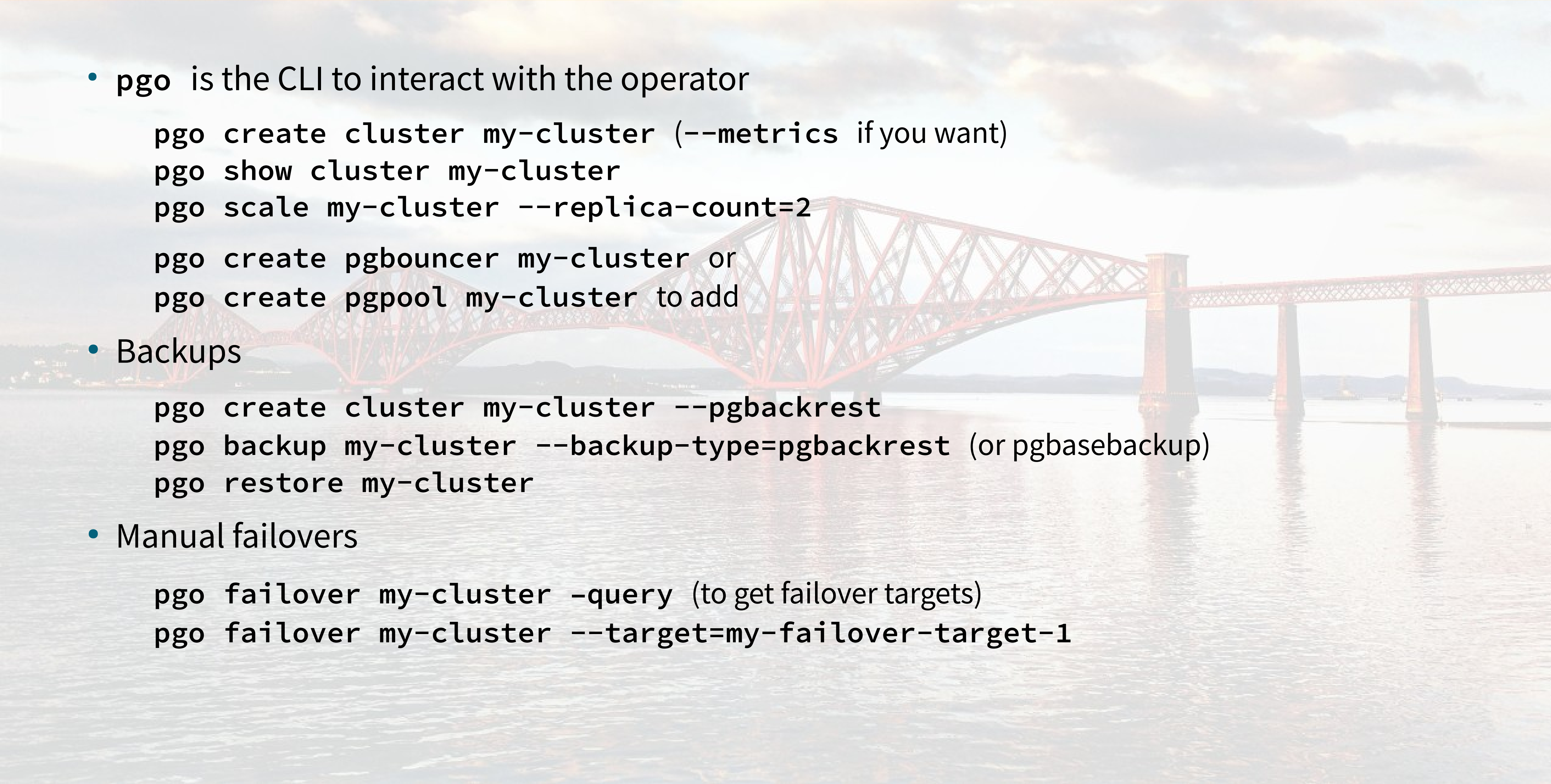
Crunchy Operator – 2



Quickstart:

- `git clone` the GitHub repo, `git checkout <tag>`
- `source examples/envs.sh`
- `make setupnamespace` creates a “demo” namespace
- `conf/postgres-operator/pgo.yaml` holds the configuration
- `make installrbac` Creates RBAC resources and keys
- `make deployoperator`

Crunchy Operator – 3: pgo



- **pgo** is the CLI to interact with the operator

  `pgo create cluster my-cluster` (`--metrics` if you want)
  `pgo show cluster my-cluster`
  `pgo scale my-cluster --replica-count=2`

  `pgo create pgbouncer my-cluster` or
  `pgo create pgpool my-cluster` to add

- Backups

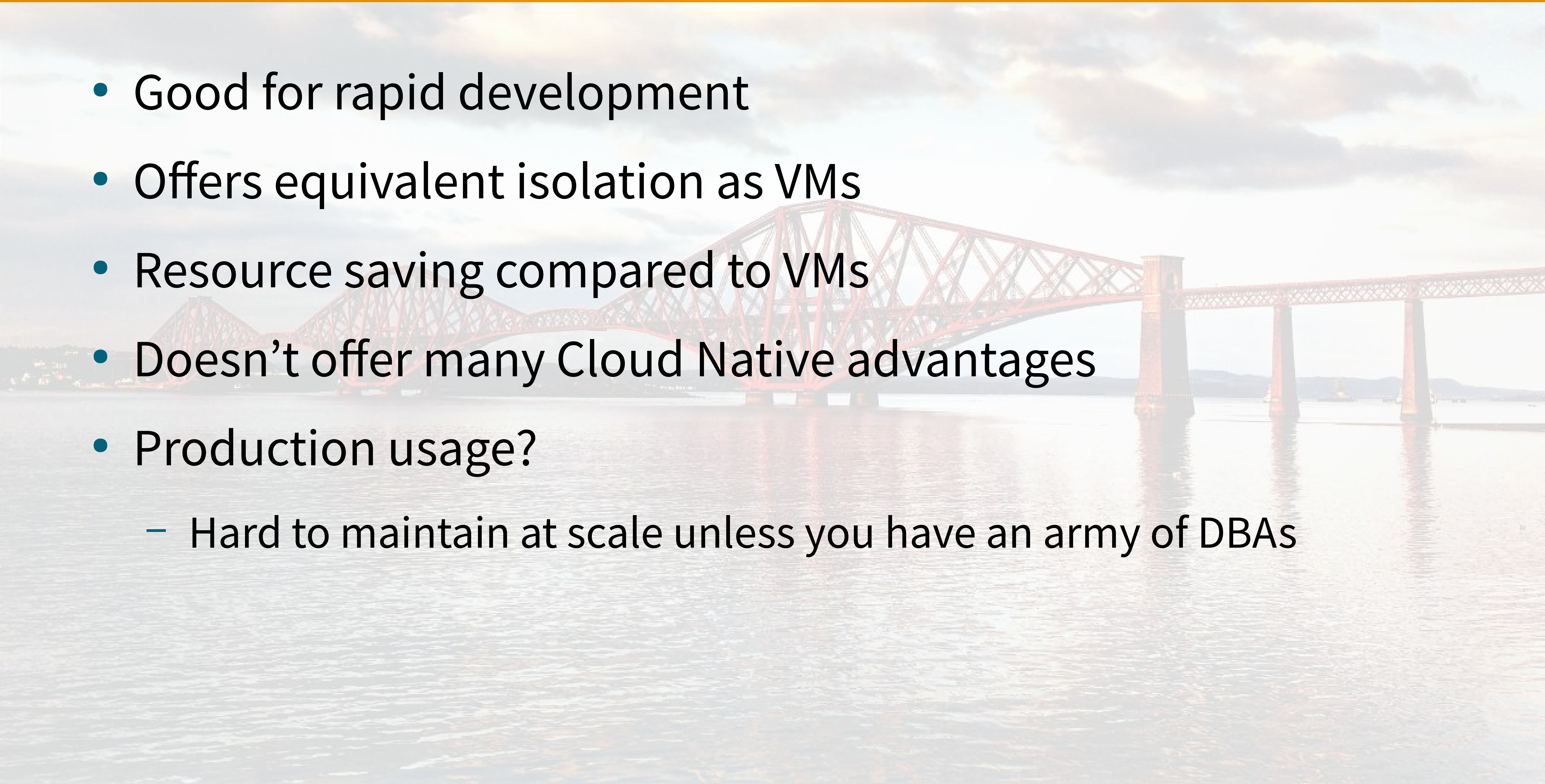
  `pgo create cluster my-cluster --pgbackrest`
  `pgo backup my-cluster --backup-type=pgbackrest` (or pgbasebackup)
  `pgo restore my-cluster`

- Manual failovers

  `pgo failover my-cluster -query` (to get failover targets)
  `pgo failover my-cluster --target=my-failover-target-1`

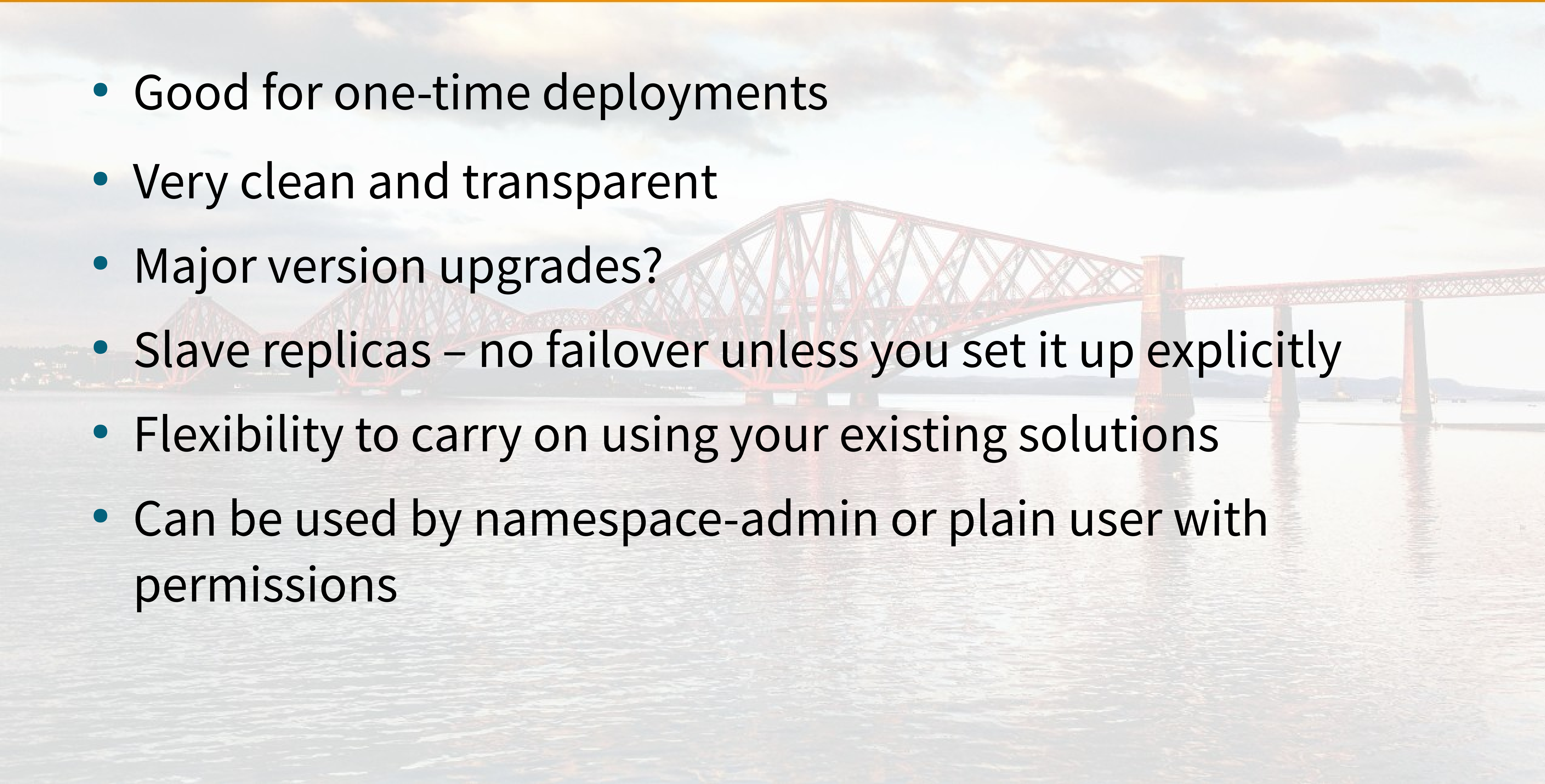
V. Observations

Observations – 1: Deploying by hand



- Good for rapid development
- Offers equivalent isolation as VMs
- Resource saving compared to VMs
- Doesn’t offer many Cloud Native advantages
- Production usage?
  - Hard to maintain at scale unless you have an army of DBAs

Observations – 2: Helm Charts

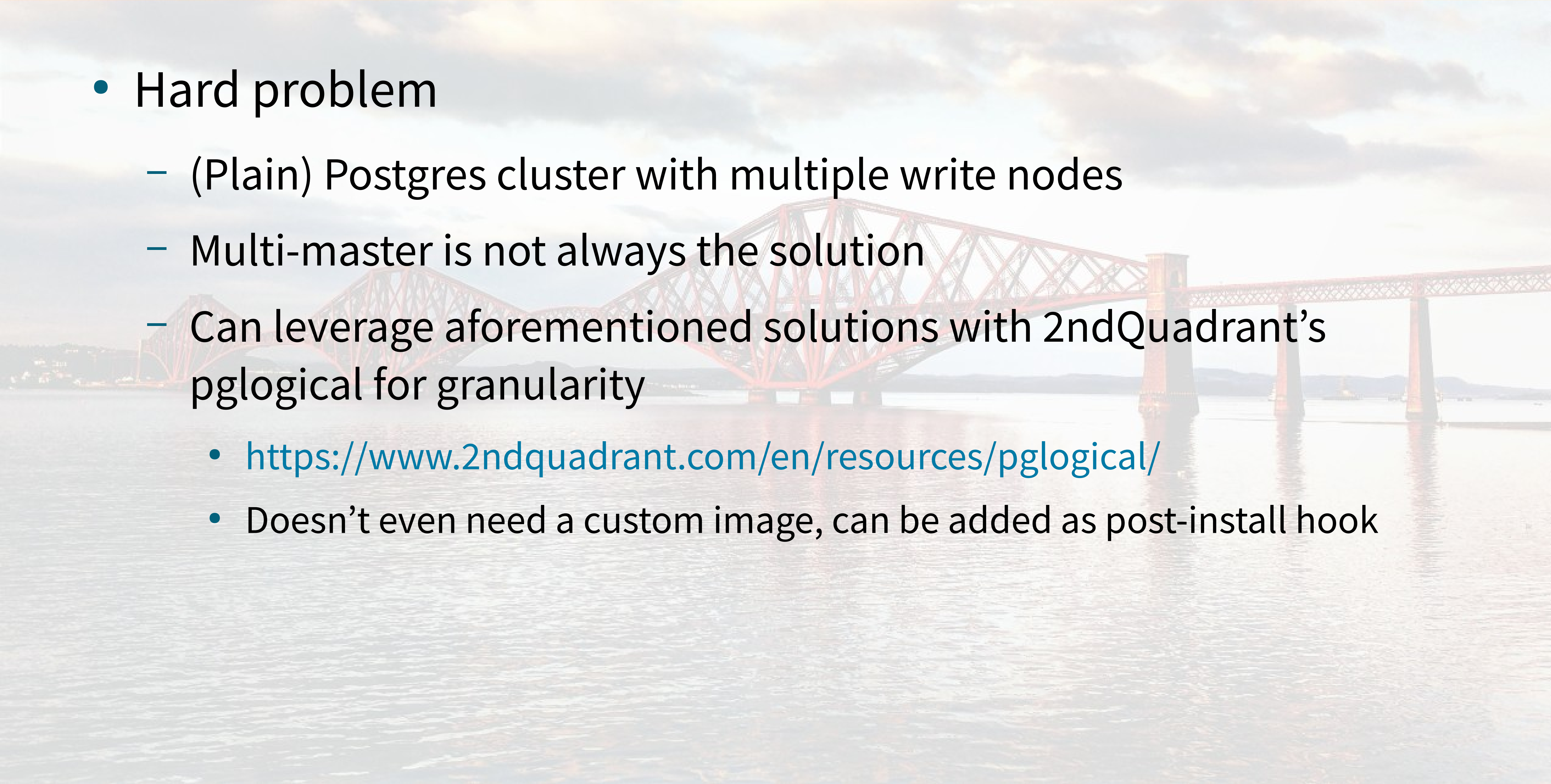


- Good for one-time deployments
- Very clean and transparent
- Major version upgrades?
- Slave replicas – no failover unless you set it up explicitly
- Flexibility to carry on using your existing solutions
- Can be used by namespace-admin or plain user with permissions

Observations – 3: Crunchy Operator

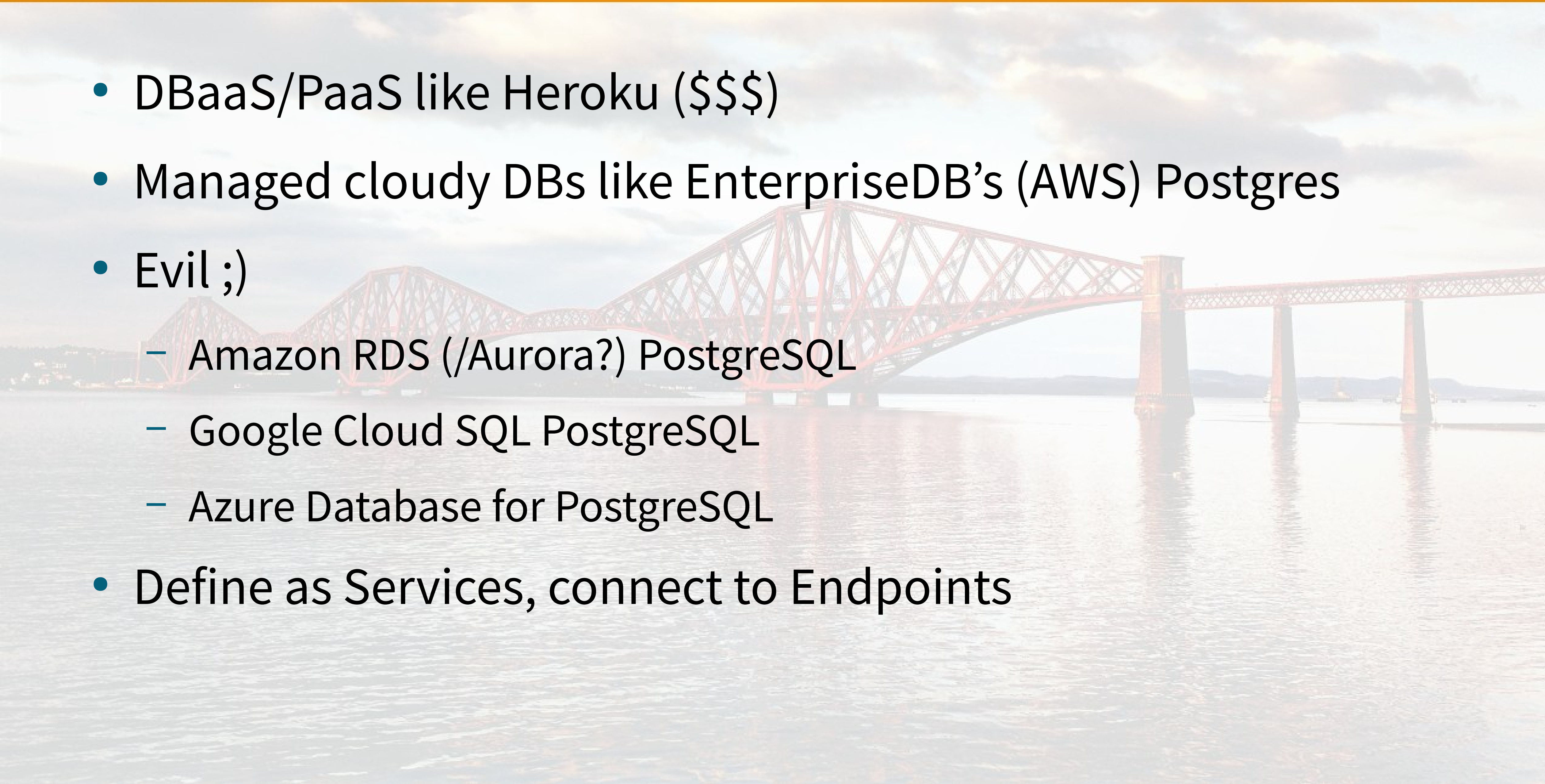
- All-in-one solution, Postgres as an application
- Makes many tasks easy via CLI and automates others
- You need RBAC and cluster-admin permissions for creation of CRDs
  - Kubernetes does not support namespaced CRDs :(
  - https://github.com/kubernetes/kubernetes/issues/65551
- Under heavy development – perhaps not ideal for production?
  - But so is Kubernetes :/

Observations – 4

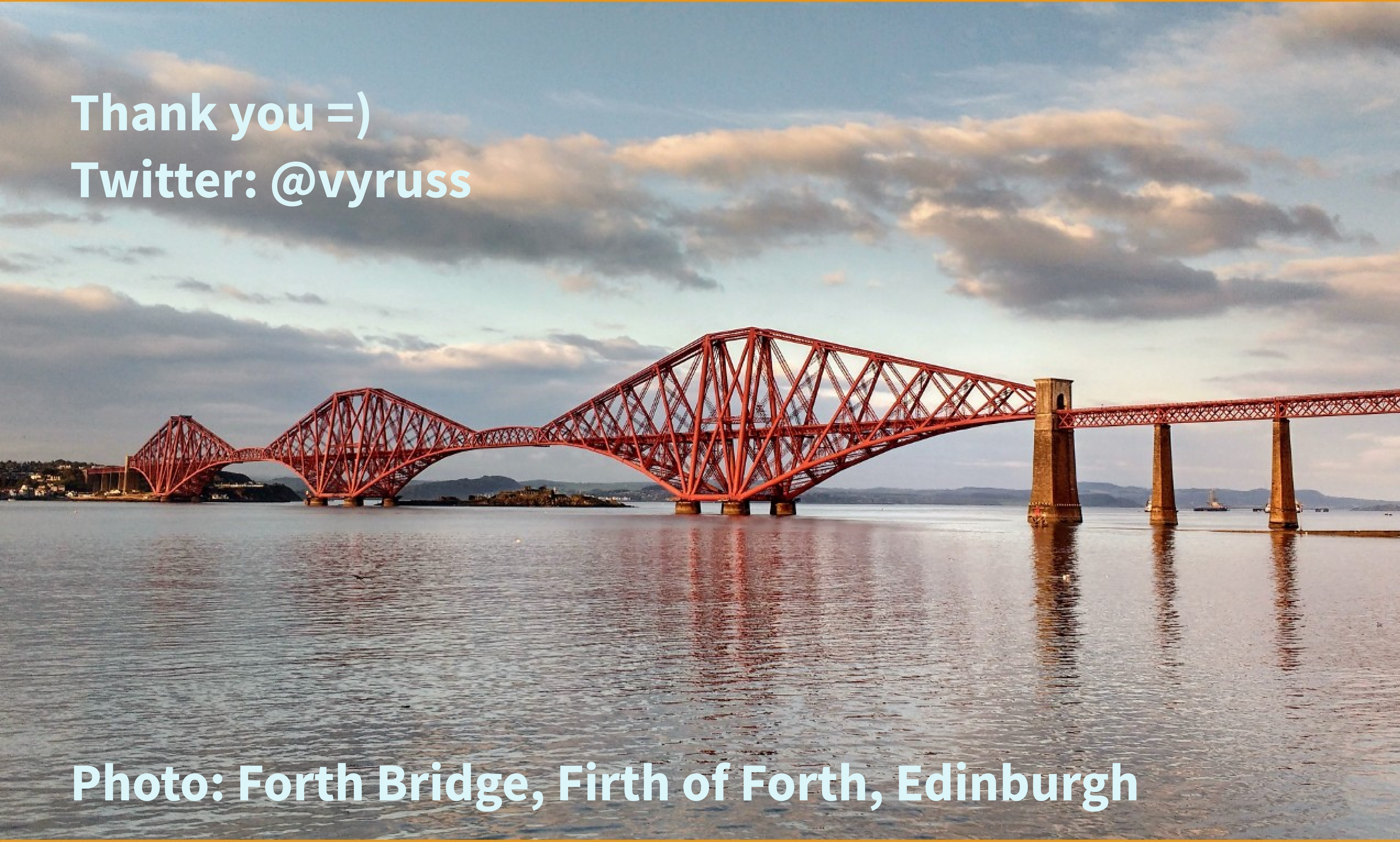


- Hard problem
  - (Plain) Postgres cluster with multiple write nodes
  - Multi-master is not always the solution
  - Can leverage aforementioned solutions with 2ndQuadrant’s pglogical for granularity
    - https://www.2ndquadrant.com/en/resources/pglogical/
    - Doesn’t even need a custom image, can be added as post-install hook

Alternatives?



- DBaaS/PaaS like Heroku ($$$)
- Managed cloudy DBs like EnterpriseDB’s (AWS) Postgres
- Evil ;)
  - Amazon RDS (/Aurora?) PostgreSQL
  - Google Cloud SQL PostgreSQL
  - Azure Database for PostgreSQL
- Define as Services, connect to Endpoints



Thank you =)

Twitter: @vyruss

Photo: Forth Bridge, Firth of Forth, Edinburgh