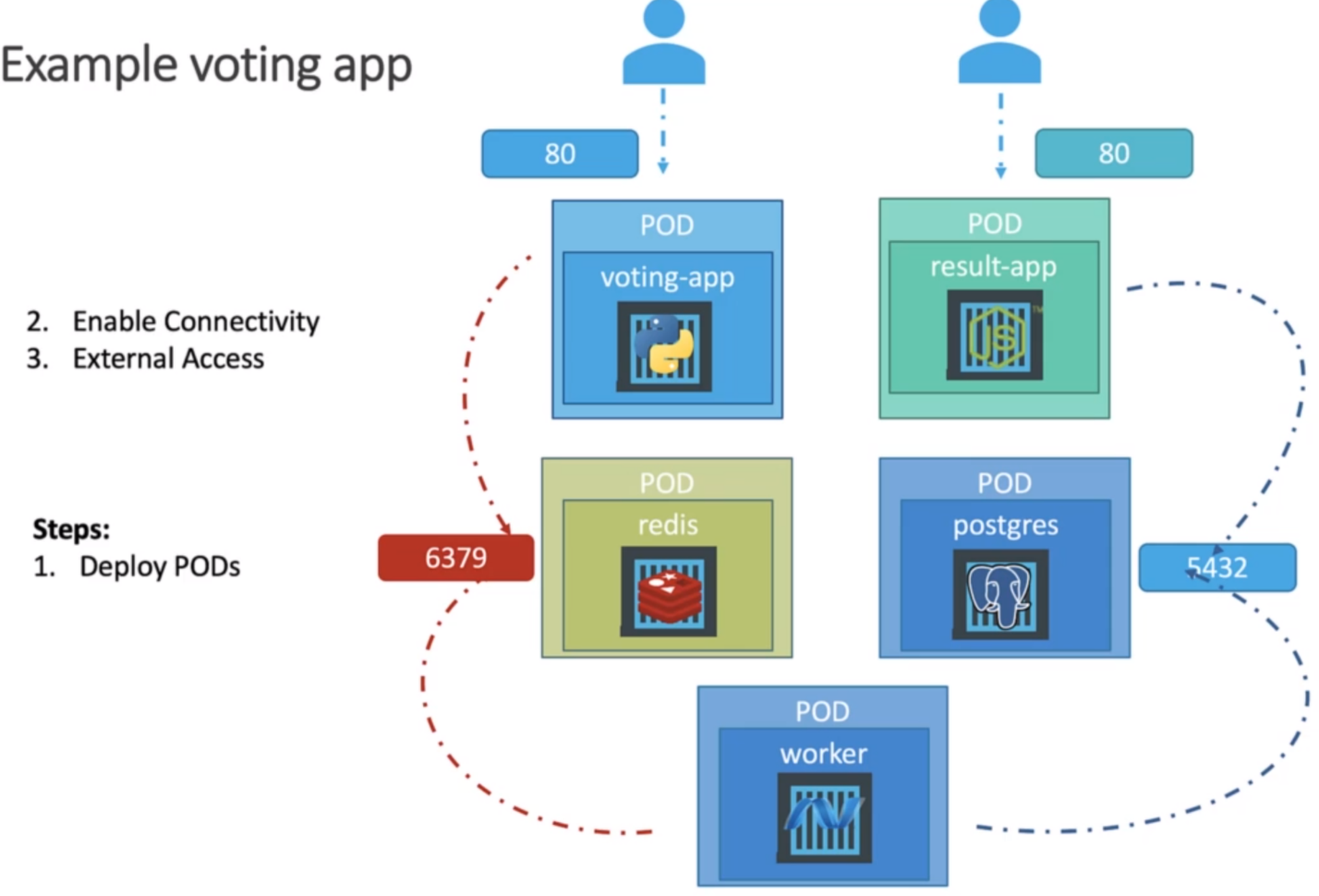
So, we just saw how the voting application works on Docker. Let’s now see how to deploy it on Kubernetes. For that our goal is to deploy the application containers on a Kubernetes cluster, and then enable connectivity between the containers so that the applications can access each other and the databases; and then enable external access for the external facing applications which are the voting-app and the result-app so that the users can access the web browser. How to do this?

We know we cannot deploy containers directly on Kubernetes. The smallest object that we can create on a Kubernetes cluster is a pod. So, we must first deploy these applications as a pod on our Kubernetes cluster, or we could deploy them as Replica Sets or Deployments. For the sake of simplicity, we will stick to the pods in this lecture. Once the pods are deployed, the next step is to enable connectivity between the services. It’s important to know what the connectivity requirements are.

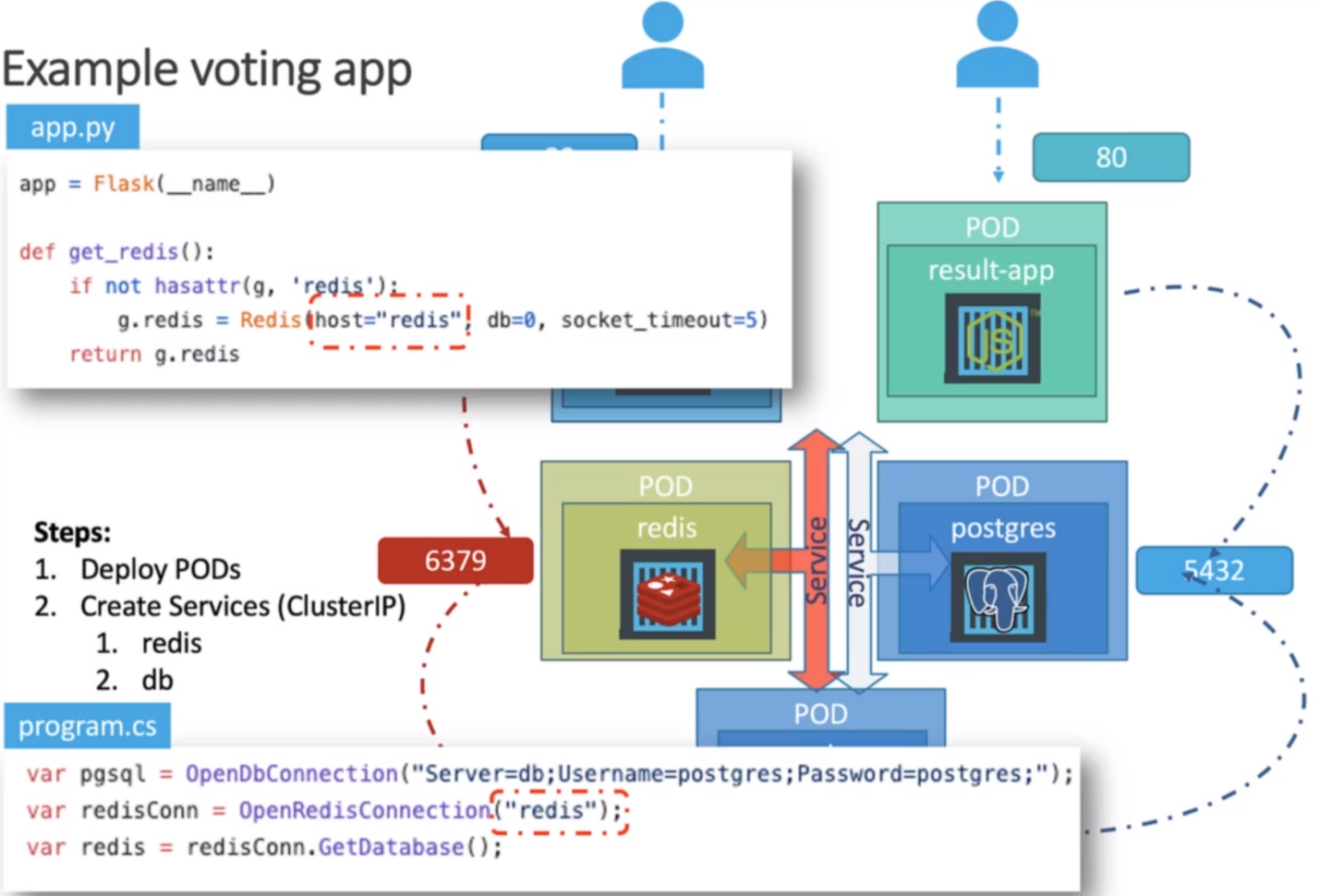


From the above arrows, you can check the different apps accessed by the different applications/users. The voting-app is accessed by the external user and in turn it accesses the Redis cache to store the vote count. Redis is being accessed by worker to save the count in database. Database is accessed by the result-app. Therefore, every app has two-way interaction but only worker app has one way dependency as could be seen from the above image as well.

Also, the worker app has no service because it’s just a worker and it’s not accessed by any other services. So, how you make one component accessed by another? How do you make Redis database accessible by the voting app?

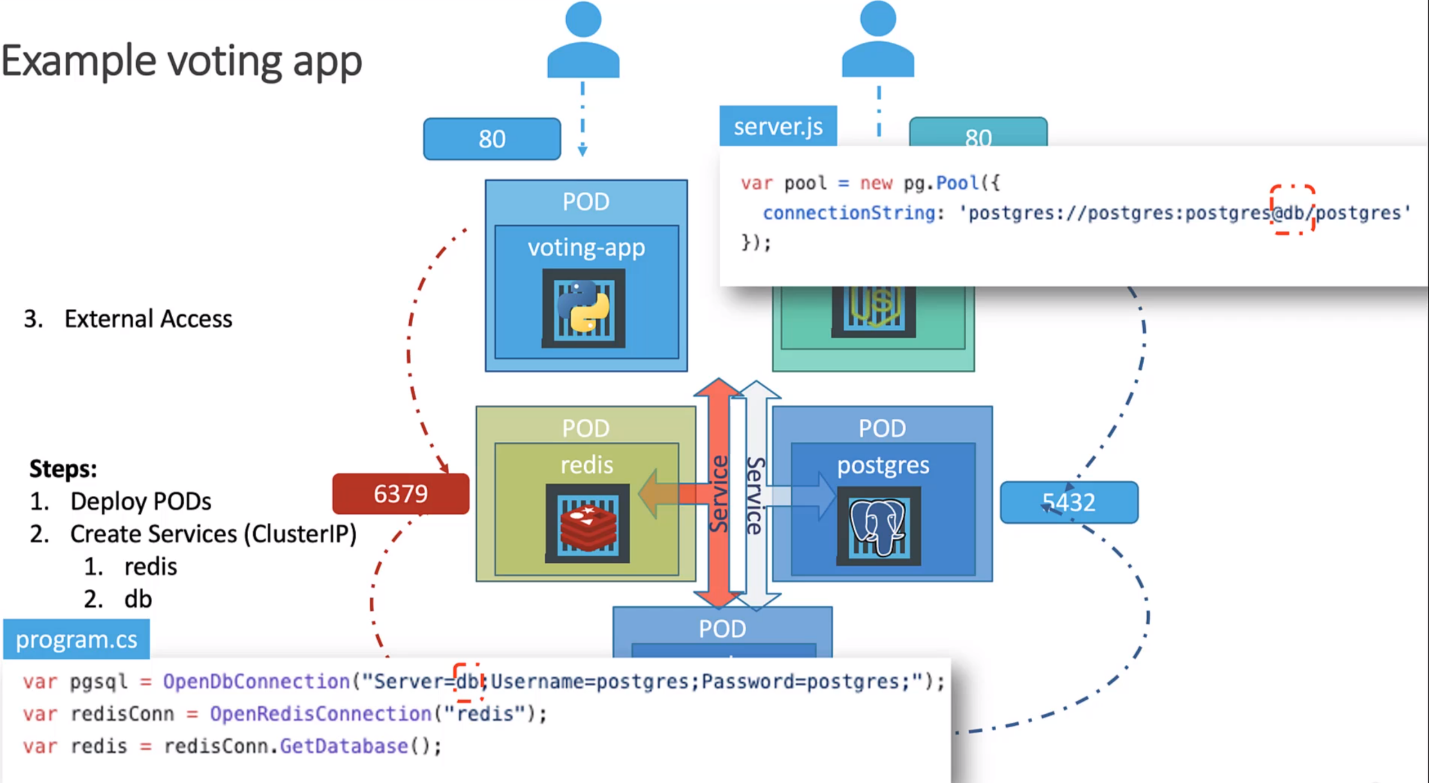
The right way to do is to use a service. We know a service can be used to expose an application to other application or users for external access. So, we will create a service for Redis app that could be accessed by the voting-app and the worker app. We will call it Redis service and it will be accessible anywhere within the cluster by the name of the service. Why name is important?

The source code within the voting-app and the worker app are hardcoded to point to a Redis database running on a host by the name Redis. So, it’s important to name your service as Redis so that these applications can connect to the Redis database. This is not a best practice to hard code stuff like this within the source code of an application. Instead, you should be using environment variables or something, but for the sake of simplicity, we will just follow this application as it is developed.

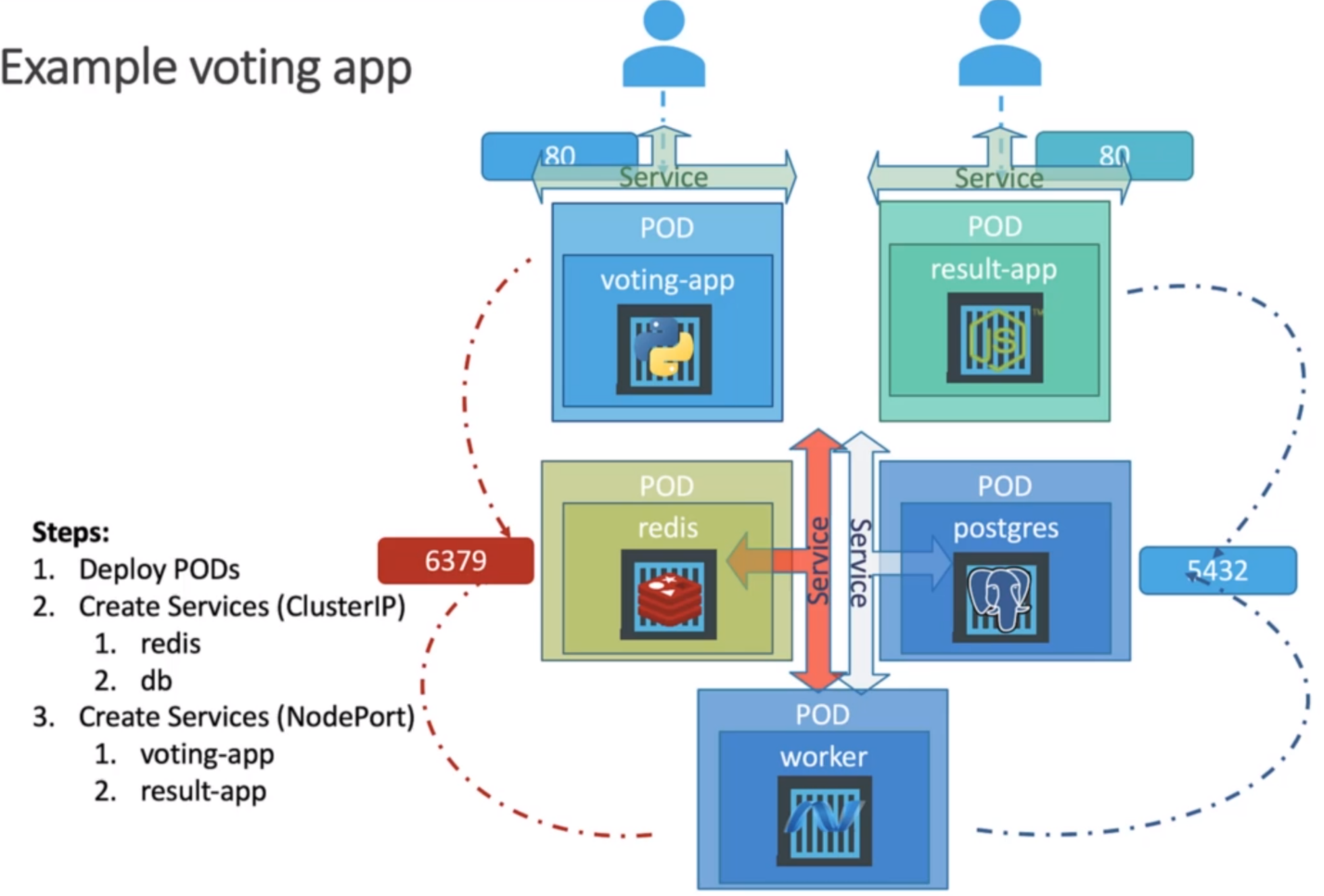


These services are not to be accessed outside the cluster, so they should just be of type ClusterIP. We will follow the same approach of creating a service for the Postgres SQL pod. What should we name the Postgres SQL service?

If you look at the source code of the result-app and the worker app, you will see that they are looking for a database @db, so the service that we create for Postgres SQL should be named DB. While connecting to the database, the worker and the result-app are passing a username and password to connect to the database, both of which are set to Postgres. So, when we deploy the Postgres DB Prod, we must make sure that we set the credentials for it while creating the database.



To enable the external access, we have to use the type NodePort. So, we create a service for voting-app and the result-app and we have to set the type as NodePort.



Therefore, we have created 5 pods and 4 services. But the question remains, why the worker does not require a service?

A service is only required if the application has some kind of process or database service or web service that needs to be exposed, that needs to be accessed by others, in this case that’s not true for the worker app.