V9 in Human Language

Micro Services

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are:

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

We decided to use Microservices for V9 for all these reasons above.

Docker

Docker and all its goodies

Docker in general

Docker is used to create, deploy and run applications (such as the Identity server en WebMVC) as containers.

Docker Images

A Docker image is like a read-only snapshot of an application, used to create containers. Each service in V9 will have their own image (Account.API, WebMVC, etc).

Imagine that you have created your containers from the images. Now, it gets a little bit more complicated when you are working with a bigger applications that has 10 to 10000 containers that should run and also where some containers will have multiple copies running simultaneously. This is where an orchestrator comes in.

Kubernetes

Kubernetes and all its goodies

Kubernetes is the most popular orchestrator (made by Google). Kubernetes (k8s) is in charge of orchestrating all of your containers. For example, if you indicate that the WebMVC should have 3 containers running, Kubernetes makes sure that there are always 3 running. If one of them fails for some reason, a new one should be started. Kubernetes also makes sure that you don't have to worry about the container IP addressees when communicating with one of them. Because there are 3 containers, there are also 3 different IP addresses. And because one of these containers can fail and a new one has to be started, it means that the new container will have a new IP address. So you can see why things can quickly get out of control, however, Kubernetes also handles the networking. It makes sure that you can communicate with ONE IP or DNS and reach one of the 3 containers (preferably not one that is busy failing or starting up) via some sort of scheme such as round robin, load balancing, etc.

Pods

Kubernetes works with resources and everything is just seen as a resource. The smallest resource of an application is called a Pod. The Pod can be seen as the computer that the application is run on and the Pod is then running a container. The pod can also run more than one container at a time. This practice is usually used for logging or proxying where you run a logger application with your main application in the same pod (on the same computer). This logger application is called a ***sidecar***. Usually, we only run one container in a pod. If you want 3 instances of your application, you will create 3 pods - usually via Deployments.

Containers within a pod can communicate with one another through localhost (since they are on the same "computer"). However, Pods can only communicate with one another through their cluster IPs (This is the IPs they have in the cluster). So the entire Kubernetes cluster can be seen as one giant private network (such as a company with several computers within it). However, previously we said that the IPs can get out of control, since they can change any time and we don't always know what the new IPs will be. That's why Kubernetes uses DNS (usually provided through the CoreDNS module) via Services. Pods can thus communicate with one another through Services.

PS: The WebMVC can essentially only serve one person at a time, it just happens so quickly that one thinks it's serving all at once, but when usage gets more one will realize that this is not the case. So by running 3 containers, the load is essentially split between them (load balancing).

HELM

Helm and all its goodies

Kubernetes uses a configuration file (YAML file) to contain all sorts of settings that a container needs to function on a specific deployment. For example, the WebMVC app needs to know where the IdentityServer is located. You can place all these type of settings in the associated YAML file, but now you have to change something such as deploying it on another premise where the IdentityServer URL is different, since it has to be a public URL (the user should also be able to access the URL, not just the services in the cluster). You now have to change the URL in each YAML file of all 500 services... sounds tedious right?

This is where HELM comes in - it basically allows you to use the YAML files like templates. So instead of indicating that the URL should be "https://www.identity.signifyhr.co.za", you can rather indicate it as ".Values.IdentityUrl". You essentially changed the value into a variable. All 500 YAML files can now reference this one variable and you can set the value in one location for the specific release.

Chart

A *Chart* is a Helm package (the location to set the values). It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster. So let's say we have to deploy our application on ARM for example, we can locate the correct Chart for the version that the client wants and tweak a few settings then install or upgrade the Chart on ARMs Kubernetes cluster.

Revision #20
Created 24 July 2020 11:43:24
Updated 23 November 2022 12:54:13