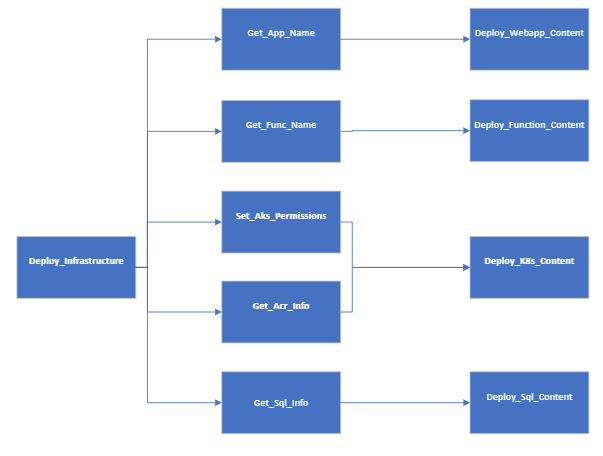
**Estructura de Pipelines**

Principalmente el sistema de los pipelines se divide en 5 pipelines. Un pipeline principal para crear los recursos y desplegar los contenidos y cuatro pipelines de contenido para los cuatro recursos (Aks, Azure Function, Azure web app, Azure sql), para sí hace falta modificar el contenido de un de los recursos, entonces de esta manera no hará falta volver a desplegar toda la infraestructura de cero.

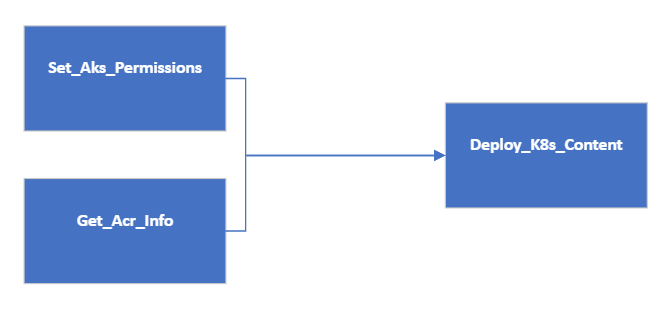
**Principal-pipeline**

Este es el pipeline principal, este pipeline es el pipeline responsable de crear todos los recursos, i también desplegar todos los contenidos de cada recurso, este pipeline llama a todos los stages que están definidos dentro del proyecto, el fichero que contiene este pipeline es el fichero principal-pipeline.yml, y tiene un trigger programado para lanzarse si se cambia un de los ficheros principal-pipeline.yml, main.bicep o algún fichero dentro del el directorio bicep-templates.



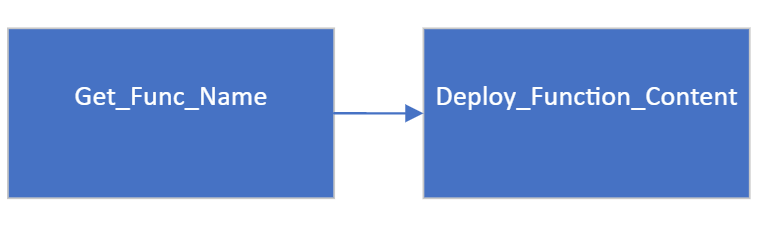
**Aks-pipeline**

Este pipeline es el pipeline responsable de desplegar los contenidos dentro del aks, este pipeline contiene el stage Set\_Aks\_Permissions, Get\_Acr\_Info y Deploy\_K8s\_Content y su trigger se lanza en función si se cambia el fichero aks-pipeline.yml o algún fichero del directorio content/aks-content/.



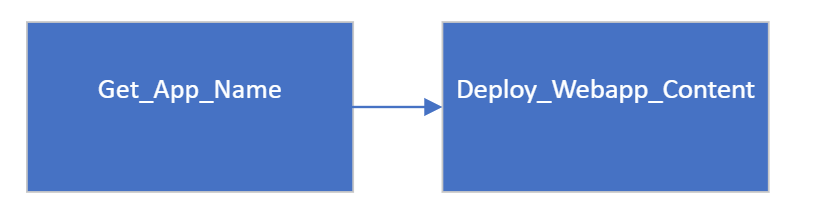
**Azure-function-pipeline**

La funcionalidad de este pipeline es desplegar el contenido dentro del azure function y este pipeline solamente contiene dos stages que se llamen Deploy\_Function\_Content y Get\_Func\_Name, y el trigger de este pipeline se lanza si se cambia el fichero function-pipeline.yml o algún fichero dentro del directorio content/ function-content/.



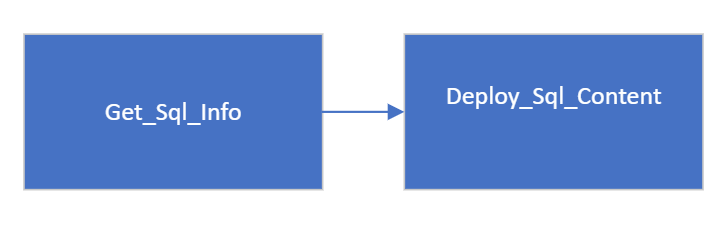
**Azure-webapp-pipeline**

La funcionalidad de este pipeline es el desligué del contenido del azure webapp, este pipeline contiene dos stages que son Get\_App\_Name y Deploy\_Webapp\_Content, el pipeline está definido dentro del fichero webapp-pipeline.yml y su trigger se lanza si se cambia el contenido del fichero webapp-pipeline.yml o los ficheros dentro del directorio content/webapp-content/.



**Azure-sql-pipeline**

Finalmente tenemos este pipeline que es básicamente un pipeline para desplegar la base de datos que como los otros pipelines de contenidos tiene dos stages que se llamen Deploy\_Sql\_Content y Get\_Sql\_Info, el pipeline está definido dentro del fichero sql-stage.yml y se lanza su trigger si se cambia el fichero sql-pipeline.yml o los ficheros del directorio content/sql-content/.



**Estructura de Stages**

Los stages son la unidad de trabajo más grande que se puede definir en los pipelines de azure, también dentro de los stages podemos definir más de una tarea que se pueden compilar en diferentes entornos.

En nuestro proyecto los stages están definidos en ficheros independientes de los ficheros que están definidos los pipelines y de esta manera ganamos la aventaja de que en caso de que más de un pipeline necesiten el mismo stage, entonces no nos hará falta definir el mismo stage más de una vez, sino solamente con llamar el stage desde los diferentes pipelines ya lo tendremos.

A continuación vamos a detallar en cada uno de los stages en nuestro proyecto.

Get\_Acr\_Info

Este stage se usa en los pipelines principal-pipeline y Aks-pipeline, se llama justamente antes de llamar el stage Deploy\_K8s\_Content.

La funcionalidad de este stage es traer el nombre y la url del acr, porque posteriormente el stage Deploy\_K8s\_Content, puede usar esta información para subir las imágenes al acr y también conectarse el aks con el acr.

Este stage consigue su tarea partiendo del sistema de tags que se define previamente para todos los recursos en el momento del despliegue de este proyecto, en los ficheros de bicep.

Set\_Aks\_Permissions

En este stage se da los permisos necesarios al aks para que el exportador de métricas de azure pueda tener los permisos suficientes para poder acceder a las métricas.

Los permisos que se dan en azure es a nivel de clúster, y el role es un role para leer las métricas desde azure, en este stage hay una variable que se llama useAksMonitoring que tiene que estar a true para que se ejecute este stage que si no, este stage se salta.

Este stage es necesario para los pipelines principal-pipeline y Aks-pipeline

Deploy\_K8s\_Content

Este stage se llama desde los pipelines principal-pipeline y Aks-pipeline, lo que hace este stage es primero conectarse al acr y subir la imagen del proyecto del cliente y otra imagen para el exportador de métricas que utiliza el prometheus.

Finalmente despliega cinco proyectos usando helm que son el controlador de ingress- nginx, la aplicación del cliente, grafana y prometheus, elasticsearch y el desligue del exportador de métricas de azure para prometheus.

Este stage tiene dos dependencia al stage de Set\_Aks\_Permissions y Get\_Acr\_Info.

Get Func Name

La funcionalidad de este stage, es conseguir el nombre del azure function, ya que este nombre se elige de forma aleatoria, y es importante para el stage Deploy\_Function\_Content, para poder subir el contenido dentro del azure function.

El stage localitza el recuso del azure function, usando el sistema de tags aplicados a todos los recusos dentro de este proyecto.

Deploy\_Function\_Content

En este stage hay más de una tarea definida que básicamente lo que hace el stage es conectarse al azure function y preparar todo el entorno y instalar los paquetes necesarios que define el usuario dentro del fichero requierments.txt y finalmente comprime todos los ficheros y los sube dentro del azure function

Este stage tiene una única dependencia al stage Get\_Func

Deploy\_infrastructure

Este es el stage principal y no hay ni una stage que se puede ejecutarse sin que previamente este stage se halla ejecutado y que no haya dado errores dentro del Principal-pipeline, todas las otras stages tienen dependencia a este stage aun que no sea dependencia directa.

Este stage es responsable de desplegar todos los recursos usando el fichero main.bicep y las plantillas dentro de la carpeta bicep-templates que también están definidos en bicep.

Get\_App\_Name

En este stage basicamente se consigue el nombre del recurso del webapp, para que el stage Deploy\_Webapp\_Content, pueda localizar el webapp y pueda subir el código.

Este stage localitza el recuso del webapp, usando el sistema de tags aplicados a todos los recusos que se depliguen dentro de este proyecto.

Deploy\_Webapp\_Content

Aquí básicamente se prepara el entorno de la webapp y se hi instalar todos los paquestes necesarios, y finalmente se comprime el conjunto de fichero y se sube todo a la web app.

Este stage se llama desde los pipelines Webapp-pipeline y principal-pipeline.

La dependencia que tiene este stage es dependencia directa al stage Get\_Sql\_Info.

Get\_Sql\_Info

La funcionalidad de este stage es conseguir el nombre y el url del servidor sql, para que después el Deploy\_Sql\_Content, pueda subir la base de datos dentro del servicio de sql.

Deploy\_Sql\_Content

Este stage es responsable de subir el contenido en la sql service.

Lo que hace este stage básicamente es, abrir el cortafuego para poder subir la base de datos y después sube el contenido de la base de datos y finalmente vuelve a cerrar el cortafuego.

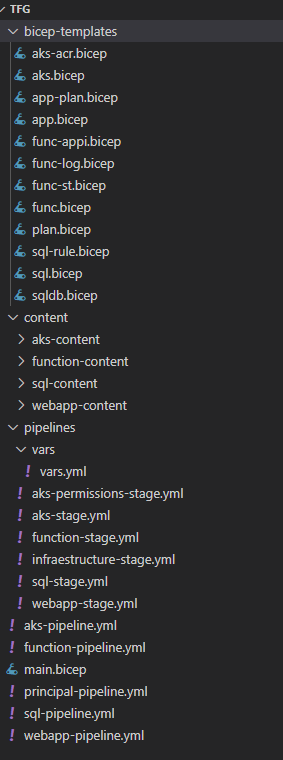
Este stage tiene dependencia directa al stage Get\_Sql\_Info

**Azure Repository Structure**

En el repositorio de azure nosotros tenemos todos los ficheros para nuestro proyecto, básicamente allá hay todo, y nosotros hemos decidido de tirar por esta vía por dos razones principales:

1. Para proteger nuestros ficheros y datos para no perderlos.
2. Ya que usamos el cloud de azure, nos resultara mucho más fácil usar azure repositorio que otro repositorio, ya que también usamos azure pipelines.

En el directorio root de nuestro repositorio lo que encontramos son tres carpetas y seis ficheros, que a continuación vamos a explicar de que consta cada uno



**Los seis ficheros son**

Primero, los cinco ficheros que definen los cinco pipelines que necesitamos para el diseño de nuestro sistema de integración continua (principal-pipeline.yml, aks-pipeline.yml, function-pipeline.yml, webapp-pipeline.yml, sql-pipeline.yml) y main.bicep que es el fichero principal que llama a las diferentes plantillas para desplegar la infraestructura de recursos dentro de azure.

**Los tres directories son los siguientes**:

bicep-template

En este directorio hay 11 ficheros, que contienen los diferentes plantillas de los diferentes recursos usados para desplegar la infraestructura, las plantillas están definidas en bicep, y todas se llaman desde el main.bicep.

  A continuación tenemos la lista de los ficheros y el contenido de cada fichero

* aks-acr.bicep: Este fichero contiene la plantilla que se usa para crear el registro de contenedores que usar el aks para obtener las imágenes para los pods.
* aks.bicep: Aquí está definida la planilla del aks para desplegarse el cluster.
* app.bicep: Aquí están el despliegue del webapp.
* func-appi.bicep: Este fichero se usa para el despliegue del application insights que se usa con el azure function.
* func-log.bicep: Este fichero se usa para desplegar el log analytics que usa el azure function.
* func-st.bicep: Aquí se crea el storage account que se usa para que el azure function puede guardar sus datos.
* func.bicep: Este es la plantilla que se usa para el azure function.
* plan.bicep: Este fichero se usa para crear dos app service plan uno para el azure function y el otro para el webapp.
* sql-rule.bicep: Este fichero es para crear los accesos en el firewall del servidor sql que por defecto tiene todos los accesos negados.
* sql.bicep: Aquí se crea la base de datos dentro del servidor sql.
* sqldb.bicep: Esta plantilla se usa para crear el servidor sql que contiene la base de datos.

content

Este directorio contiene 4 subdirectorios aks-content, function-content, sql-content, webapp-content, que cada uno de estos cuatro subdirectorios contiene el contenido que se usa en los recursos (azure kubernetes service, azure function, azure webapp, azure sql).

aks-content: Aquí hay 6 subdirectorios, cada uno es para un componente diferente dentro del aks.

* App: Este subdirectorio se usa para desplegar la aplicación del cliente dentro del aks.
* docker-images: Este subdirectorio contiene 2 imágenes de docker una para la aplicación del cliente y otra para el exportador de métricas que usar prometheus para traer las métricas de azure.
* ingress-ngnix: aquí está definido el contenido que se usa para desplegar el controlador de ingress-ngnix.
* kube-prometheus-stack: Aquí están las definiciones de los diferentes componentes que se usen para prometheus y grafana para correr dentro del cluster.
* prometheus-exporter: Este es el subdirectorio que define el despliegue del exportador de métricas que usa prometheus dentro de azure.

Pipelines

Dentro de este directorio están definidas todas las stages que se usen y también los variables que se usen en todo el proyecto

* vars: Es un subdirectorio que contiene un fichero que se llama vars.yml que allá están todas las variables de toda el sistema.
* aks-permissions-stage.yml: Aquí está el stage que actúa para dar al clúster de k8s los permisos necesarios para el exportador de métricas de azure.
* aks-stage.yml: Es un fichero donde está definido el stage responsable de subir el contenido de aks en el mismo.
* function-stage.yml: Este stage es el responsable de subir la función definida del usuario dentro del azure function.
* infraestructure-stage.yml: Aquí está definida el stage que desplegá el código de bicep para desplegar toda la infraestructura.
* sql-stage.yml: En este fichero está definida la stage responsable de subir la base de datos en el sql service de azure.
* webapp-stage.yml: Este stage es el responsable de subir el código de la aplicación web del cliente en el webapp.
* Get-info-stage: Este fichero contiene cuatro diferentes stages, que los cuatro están definidos con el mismo código y en el mismo fichero, los cuatro pipelines definidos en este fichero, son Get\_Acr\_Info, Get\_Func\_Name, Get\_App\_Name y Get\_Sql\_Info