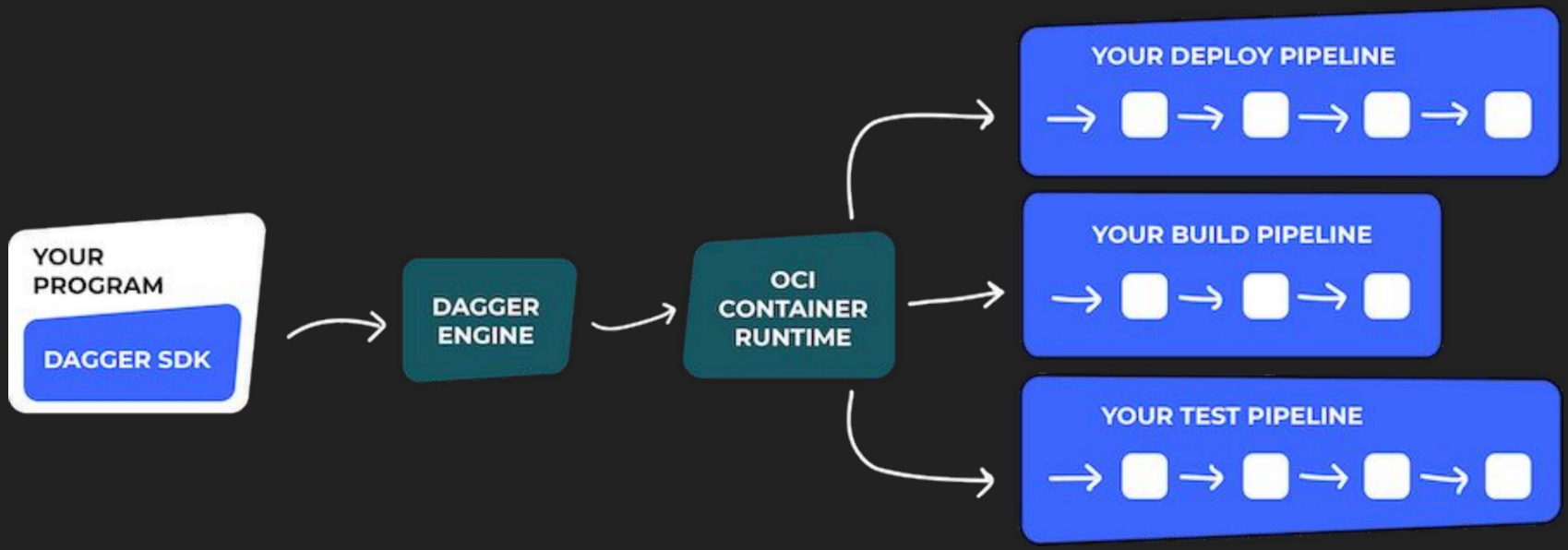

Dagger

¿Qué es?

- Motor de CI/CD que permite la ejecución de tareas en contenedores.
- Ofrece:
  - SDK en lenguajes mayoritarios (python, Javascript, go)
  - Construcción de
    - imágenes
    - ficheros
    - caches
  - Gracias al motor de contenedores puede correr en cualquier entorno que tenga un docker instalado (local, Github, Gitlab, kubernetes…)

¿Cómo lo hace?

- Emplea la api de buildkit/container para realizar acciones y exportar/importar elementos OCI.



¿Quién está detrás?

Solomon Hykes:

- Inventor de Docker
- Gurú del movimiento de la contenerización
- Tiene obsesión con la usabilidad y ergonomía de herramientas que construye

Ventajas

- Expresividad
  - Emplear la potencia del lenguaje que emplea el SDK para hacer casi cualquier tipo de tarea
    - Llamar apis
    - Bucles
    - Paralelizar/Concurrencia
    - División en funciones/módulos
- Estandarización
  - Gracias a correr todo dentro de contenedores OCI el sistema es, virtualmente, compatible con cualquier entorno donde docker pueda correr.
- Velocidad
  - La depuración de los programas es mucho más ágil
  - El poder que da la caché hace que las tareas comunes y repetitivas se puedan reciclar ahorrando gran cantidad de tiempo.

En un ejemplo

- Aplicación de nodejs
  - Instalar deps
  - Correr el linter
  - Pasar tests unitarios
  - Construir la imagen y subirla al registry
  - … En 4 m. de nodejs (14, 16, 18 y 19)

```
async function(client){

  for(const nodeEngine of ["node:14", "node:16", "node:18", "node:19"]){

    let ctx = client.container().from(nodeEngine)

      .withMountedDirectory("/app", new Client().host().directory("."))
      .withWorkdir("/app")

    // install-lint-test-build
    await ctx.withExec(["npm", "install"]).exitCode()
    await ctx.withExec(["npm", "run", "lint"]).exitCode()
    await ctx.withExec(["npm", "run", "test"]).exitCode()

    ctx = await ctx.withExec(["npm", "run", "build"])
    await.exitCode()

    // final image and publish
    await client.container().from(nodeEngine + '-slim')

      .withDirectory("/app/public", ctx.directory("/app/dist"))
      .withEntrypoint(["node", "/app/public/index.js"])
      .publish(`<REGISTRY_ADDRESS>:${nodeEngine}`)
  }


}
```

Desventajas

- API no madura (0.4.0)
- Mala documentación (incipiente)
- Faltan funcionalidades
- ¿Segura? (tiene gestión de secretos)
- Puede generar dudas en el cliente a la hora de su implantación
- Visualización de errores es complicada

¿Podemos adoptarlo en nuestros clientes?

- Experiencia piloto en smartlex-quasar (sudespacho)
  - Aplicación de frontend multiflavour
  - Necesitan un workflow de construcción/despliegue “a la carta”
  - La configuración principal de la aplicación está en el repo de código
- “Think big start small”
  - Probar con ciclos y tareas sencillas
  - Tener en cuenta que puede haber muchos cambios en la API
  - Comenzar a identificar patrones y rutinas comunes

¡Gracias!

Fundamentos teóricos

- **Containerización de function-calls**
- **Memoización**
  - **DAG (Directed Acyclic Graph)**
  - **Caché de dagger**
- **Orquestación a nivel de programa**
  - **Arranque condicional y despliegue en stacks diferentes**