Devfile support in odo

To better integrate odo with other developer tooling we need to align on one common definition of a developer workspace and application lifecycle.

Eclipse Che project already created and started using Devfile. Devfile is a YAML file that defines developer workspace in the Che environment. It is structured in such a way that it can be used also in odo.

Implementing Devfile support in odo will create a great benefit for users, as it will allow them to easily switch tools with no additional configuration. Adding support for new languages will be also much easier with devfiles, as it will be only a matter of using the right devfile template.

Implementation plan

Consuming the current version of Devfile in odo

1. Create a Golang library for parsing and reading data from Devfile
    - Devfile is versioned file so the design of a library needs to be able to handle and support multiple Devfile versions at the same time

      Parser high-level design:

      - The parser would read the provided devfile.
      - From the devfile, the parser would look for the API Version of the devfile and validate whether the API version is supported in odo.
      - The devfile schema is based on http://json-schema.org/draft-07/schema#
      - Hence, the parser would validate the integrity of the provided devfile with the devfile JSON schema of the respective API version.
      - Furthermore, the parser would try to map devfile sections (projects, components, commands, etc) to their respective Golang structs.
      - The idea is to have the same Golang structs for the devfile sections across different API versions contingent to the

condition that the sections themselves are compatible across the API versions.

- This is very similar to the way Golang structs are handled across different a new group, version, and kind (GVK) versions in Kubernetes. Scheme defines methods for serializing and deserializing API objects...
- The implementation details for this with respect to odo are being worked upon and once confirmed would be updated here.

2. Implement experimental support for reading the current version of a Devfile and bootstrap odo components from a Devfile.
    - If there is a devfile.yml and no .odo/config.yaml `odo push` will use Devfile.yaml and create necessary resources in the Kubernetes cluster.
    - Before devfile has support for a category of commands (application lifecycle commands like run, build, test, etc.) we could just add flags to odo push that will allow specifying the name of the command that should be used for build and run, as currently, those are only two commands that odo cares about.
      For Spring PetClinic Sample Application it would look like this: `odo push --run-command="maven build" --build-command="run webapp"`.
      This will be only a temporary measure until we can mark commands directly in devfile.
    - The best approach would probably be to use "fat" pods. The component will consist of a single Pod typically with two containers - "build" container and "run" container. Both containers will share a volume with the source code (emptyDir or PVC?)
    - **Build container** will use the image (docker image) based on a component field in build command definition in devfile. The main command should be something that just sleeps indefinitely. Only ad-hoc commands (build command) will be executed in this container.
    - **Run container** will use the image (docker image) based on a component field in run command definition in devfile. The main command will be the process that can control other processes. We could reuse Go implementation of the SupervisorD as it is currently used in odo s2i flow.

The SupervisorD[1] will have one service configured. This service will run a “run” command based on a definition in devfile. Thanks to the SupervisorD approach, we don’t need to keep the connection to the cluster in order to have the application up and running

- The `odo push` flow when using the devfile.yaml
    i. Create a “fat” pod with the run and build container as defined above, if it already doesn’t exist.
    ii. Sync files from local directory to $CHE_PROJECTS_ROOT (/projects by default) directory in build container. It actually doesn’t matter which container is used, as both should have shared volume mounted in this location
    iii. Execute build command in build pod and wait for it to finish
    iv. Invoke command that will restart the run service in the run container. (`supervisord ctl restart run`)

Extending Devfile format

Work with folks from the Che side to extend Devfile format with additional information that is required by odo.

- Add a command (like build, run)
- URI > Ingress > /Routes support
- Storage support
- Linking multiple components (applications)
- Bootstrapping services from OperatorHub and linking it to the component (application)

Implement odo operations on top of a Devfile

- Implement commands that will modify Devfile.yaml. Uses shouldn’t need to touch Devfile.yaml, everything should be handled by odo commands
  For example:

[1] "Supervisor/supervisor: Supervisor process control ... - GitHub." https://github.com/Supervisor/supervisor. Accessed 13 Mar. 2020.

- odo create url - instead of modifing .odo/config.yaml it will add all necessary information into the Devfile.yaml
- odo delete url - will delete url information from Devfile.yaml

Convert old style odo component definitions to Devfile

Implement logic that will convert old S2I based LocalConfig definition (.odo/config.yaml) to devfile.yaml