Devstack 2.0

Scope and Context

Devstack is the centralized set of docker images, make files, and bootstrap scripts used to set up an environment that engineers can run locally to develop edX services in a consistent manner from engineer to engineer.

The following pain points have been identified in our current development environment:

- Operational concerns like schema migrations, data seeding, and configuration are opaque and not consistent between applications.
- The centralized and unversioned nature of the devstack means that changes made by one team can cause unintended knock-on effects for other teams
- The operations required to setup devstack take a long time even when someone knows exactly what to do. Longer if there is trial and error involved for engineers who are new to interacting with the stack.
- The system resources required to run the stack are more than some edX issued high-end laptops can handle.
- Dockerfiles and corresponding images are opaque to developers and challenging to change/troubleshoot.

Some of the challenges listed above are due to the design of the devstack development environment itself. Others are inherent in the design of the applications. Both will need to evolve to solve for all of the most common pain points.

Outcomes:

The development environment used to enhance edX applications has the following properties:

- Starts up quickly and uses few enough resources that developers can run it with common development tools (e.g. an IDE) on the lowest-common-denominator development laptop.
- Engineers can efficiently change many different services without having to spend a lot of time learning about each one
- Schema consistent with production **
- No manual data entry required for standard, reasonably production-like, configurations
- Path for reproducing data for custom configurations
- Pattern for developing that includes path for private/custom dependencies

Technical Vision:

Each codebase defines its own development environment based on its dependencies. Both the codebase under development and its dependencies have a consistent interface for start up and common interactions (like tailing the logs). This reduces the footprint, blast radius of changesets, and number of branching paths to support different environment setups.

A base schema snapshot in line with production can be downloaded and used, eliminating the need to step through migrations that have already been performed in production. Similarly, rather than every developer needing to execute a provision script, this devstack should provide some basic development seed data as well as an interface for adding custom local data consistently.

The applications can take their configuration as code to quickly put the app into a particular reproducible configuration.

How we get there:
In a distributed devstack where each repo is able to define its own development environment, the squad that owns the repo will be responsible for that development environment. There will also be shared tooling to support these development environments. Any such tooling would be owned by the SRE team. For example, jobs which create schema volume snapshots of our production databases.

This is the proposed **definition of done**** for an app to be devstack 2.0 compatible:

- Dockerfile in the root of the codebase which does not rely on ansible
    - **Initial provided by SRE. Owned by squad.**
- Build/push automation for building new images on merge
    - **Shared tooling owned by SRE.**
- Devstack docker-compose.yml in the root of the codebase
    - **Provided and owned by squad.**
- Bootstrap.sh script for setting up IDA and calling the bootstrap script for it's dependencies (if necessary)
    - **Provided and owned by squad**
- Setting ENV variables to customize images
    - **Shared tooling owned by SRE**
- Configurable with Mysql volume with production schema
    - **TBD: Start with shared tooling owned by SRE, may transition to Squad owned depending on how self-service it can be.**
- Standard set of targets that perform common actions (likely Make)
    - **Initial interface and contract provided by SRE. Owned by squad.**

** These are broad strokes and more implementation details would be provided regarding each of them before any squad would actually start work.

Challenges:
One of the biggest challenges is the fact that edx-platform is a dependency of all IDAs. However, most of them are only actually dependent upon a few django apps existing. Still open to ideas for how to approach this pragmatically.

- Bootstrap script per djangoapp?
- Let apps dependent on edx-platform write their own provisions?

- Create a stripped down edx-platform with only the most basic features (auth, student, courseoverview, enrollment, etc)

UNGOALS

- Kubernetes/minkube/k3s/etc

Proposed Execution Plan

- SPIKE
    - 3 engineers (1 SRE, 1 eSRE, 1 edx-platform-experienced engineer)
    - 1 week timebox effort
    - Driving the following outcomes for 1 repository:
        - Able to run development-ready docker container for target service and its dependencies with single command
        - From-scratch provisioning of all dependencies < 5 minutes
        - Establish pattern for running a light version of edx-platform as a container
        - Establish pattern for seeding required data in target service
        - Establish pattern for seeding required data in dependent service
- Coordinate with managers and eSREs to prioritize applying this pattern to high priority owned repositories.
- SRE team to prioritize enhancements to this model of development based on spike outputs