Overview - General

Objective

Whenever a PR job is run on the CI infrastructure (e.g. CircleCI), we want to build `angular.io` and host the build artifacts on a publicly accessible server so that collaborators (developers, designers, authors, etc) can preview the changes without having to checkout and build the app locally.

Source code

In order to make it easier to administer the server and version-control the setup, we are using docker to run a container on a VM. The Dockerfile and all other files necessary for creating the docker container are stored (and versioned) along with the angular.io project's source code (currently part of the angular/angular repo) in the `aio-builds-setup/` directory.

Setup

The VM is hosted on Google Compute Engine. The host OS is debian:jessie. For more info how to set up the host VM take a look at the "Setting up the VM" section in TOC.

Security model

Since we are managing a public server, it is important to take appropriate measures in order to prevent abuse. For more details on the challenges and the chosen approach take a look at the security model.

The 10000 feet view

This section gives a brief summary of the several operations performed on CI and by the docker container:

On CI (CircleCI)

- The CI script builds the angular.io project.
- The CI script gzips and stores the build artifacts in the CI infrastructure.
- When the build completes, CircleCI triggers a webhook on the preview-server.

More info on how to set things up on CI can be found here.

Hosting build artifacts

- nginx receives the webhook trigger and passes it through to the preview server.

- The preview-server runs several preliminary checks to determine whether the request is valid and whether the corresponding PR can have a (public or non-public) preview (more details can be found here).
- The preview-server makes a request to CircleCI for the URL of the AIO build artifacts.
- The preview-server makes a request to this URL to receive the artifact - failing if the size exceeds the specified max file size - and stores it in a temporary location.
- The preview-server runs more checks to determine whether the preview should be publicly accessible or stored for later verification (more details can be found here).
- The preview-server changes the "visibility" of the associated PR, if necessary. For example, if builds for the same PR had been previously deployed as non-public and the current build has been automatically verified, all previous builds are made public as well. If the PR transitions from "non-public" to "public", the preview-server posts a comment on the corresponding PR on GitHub mentioning the SHAs and the links where the previews can be found.
- The preview-server verifies that it is not trying to overwrite an existing build.
- The preview-server deploys the artifacts to a sub-directory named after the PR number and the first few characters of the SHA: `<PR>/<SHA>/` (Non-publicly accessible PRs will be stored in a different location, but again derived from the PR number and SHA.)
- If the PR is publicly accessible, the preview-server posts a comment on the corresponding PR on GitHub mentioning the SHA and the link where the preview can be found.

More info on the possible HTTP status codes and their meaning can be found here.

Updating PR visibility

- nginx receives a notification that a PR has been updated and passes it through to the preview-server. This could, for example, be sent by a GitHub webhook every time a PR's labels change. E.g.: `ngbuilds.io/pr-updated` (payload: `{"number":<PR>,"action":"labeled"}`)
- The request contains the PR number (as `number`) and optionally the action that triggered the request (as `action`) in the payload.
- The preview-server verifies the payload and determines whether the `action` (if specified) could have led to PR visibility changes. Only requests that omit the `action` field altogether or specify an action that can affect visibility are further processed. (Currently, the only actions that are considered capable of affecting visibility are `labeled` and `unlabeled`.)
- The preview-server re-checks and if necessary updates the PR's visibility.

More info on the possible HTTP status codes and their meaning can be found

here.

Serving build artifacts

- nginx receives a request for a hosted preview resource on a subdomain corresponding to the PR and SHA. E.g.: `pr<PR>-<SHA>.ngbuilds.io/path/to/resource`
- nginx maps the subdomain to the correct sub-directory and serves the resource. E.g.: `/<PR>/<SHA>/path/to/resource`

More info on the possible HTTP status codes and their meaning can be found here.

Removing obsolete artifacts

In order to avoid flooding the disk with unnecessary build artifacts, there is a cronjob that runs a clean-up task once a day. The task retrieves all open PRs from GitHub and removes all directories that do not correspond to an open PR.

Health-check

The docker service runs a periodic health-check that verifies the running conditions of the container. This includes verifying the status of specific system services, the responsiveness of nginx and the preview-server and internet connectivity.