Scope for the Deployment Policy and Principles

The Deployment principles and policies defined in this document applies to all services owned currently and in future by Engineering Systems team (dotnetes@microsoft.com). The specific deployment steps/process for each service should be documented as a ReadMe doc along with the service source code.

Deployment Principles

- The rollout updates and impact are known beforehand by our customers
- Deployment quality shows a continual, positive trend over time
- Key metrics are measured for each rollout and displayed on a dashboard (Rollout Score Card), and a clear rollback threshold is defined.
- Deploying multiple small updates is much more preferable to single large updates.
- No breaking changes (exceptions granted via policy)
- Rollouts themselves are frictionless, automated, and meet deployment quality goals
- Rollbacks (when triggered by clear thresholds) are also frictionless, automated, and meet quality goals
- Scenario tests are automated, accurately represent our customers, run prior to every deployment, and block deployment on failure
- All ad-hoc/exploratory testing is done by the dev (not as part of the rollout)
- Staging accurately represents production can be used for testing with confidence
- Staging is always green

It is important that we have metrics, with clear targets to measure how we're doing against these principles over time.

What is the deployment process?

Builds and Deployment for all our services are executed via AzDO Pipelines and the deployments follow the pattern described in the Deployment Process. A typical deployment involves kicking off the CI Pipeline for the services which involves Building the code, publishing artifacts, pre-deployment checks, deployment of bits that were built to the environment (staging or production), post-deployment checks. Each service or unit of deployment has its own pipeline for build and deploy.

Who owns deployments?

The deployments themselves can be done by anyone on the team, but point of accountability for deployments lies with Mark Wilkie (mawilkie@microsoft.com)

When can a rollout be done?

Typically rollouts are done once a week but the goal is to be able to at least deploy twice a week. The pre-req for a deployment to be started is to have green builds for at least 3 days in a row.

Who should be notified of the rollout?

Two days before a rollout is done, a email notification containing the release notes (significant features and bug fixes are called out) needs to be sent out to dncpartners@microsoft.com. This acts like a notice to eng services team internally, that the rollout process has started. Once the release notes have been sent out, staging/master goes into lock-down period for stabilization as mentioned in Staging always green section below.

A notification to the same alias once the rollout is complete also needs to be sent, explaining if there is any fallout from the rollout that would affect the customers.

How are we tracking issues that are encountered during a rollout?

See the sections GitHub Issue Tagging, Rollbacks and Hotfixes below.

Where are any logs regarding the rollout stored?

Pipeline runs store the logs for each deployment. History of deployments can be obtained from AzDO environments

Process of filing out rollout score card

Rollout Score Card Documentation

Policy

GitHub Issue Tagging

Every issue that arises as a result of a rollout must be filed on GitHub in dotnet/core-eng. These issues should include labels that specify the type of event and the repo in which it occurred.

- The event type labels are: **Rollout Issue** for issues that arise as a result of the rollout, **Rollout Manual Hotfix** for manual hotfixes, **Rollout**

Manual Rollback for manual rollbacks, **Rollout Downtime** for downtime that occurs as a result of the deployment, and **Rollout Failure** for deployments that need to be manually marked as failures.
  - Hotfixes and rollbacks that result in a deployment do not need to be filed on GitHub (i.e. only manual hotfixes and rollbacks should be filed)
  - Downtime issues may be automatically filed by telemetry
- Additionally, all issues must contain a label indicating the repo affected:
  - **Rollout Helix** for Helix rollouts
  - **Rollout OSOB** for OS Onboarding rollouts
  - **Rollout Arcade Services** for Arcade Services rollouts.

Rollbacks

- Rollbacks are triggered by manually kicking off a build/release pipeline to deploy the *n-1*th version. There should be minimal to no human intervention needed in rolling back a deployment. Every new deployable service/feature that is coming up needs to have a rollback story defined and implemented as part of the epic.
  - For Pipeline-based deployment, revert the change to production branch and kick-off a build and deploy pipeline.
  - For non-yaml based release pipelines, rollout the previous release.
  - Rolling back database changes can be tricky especially if data loss is incurred like in case of a added column/dropping a table. This is only scenario where they might be manual intervention needed for rollback.
- Workflow for Rollback:
  - When a service in production is discovered to be in a "downed" state (e.g. not taking work, constantly throwing errors, etc.) following a rollout, then rollback IMMEDIATELY to a previously known working deployment. Continue investigation in staging by reproing the failure.
  - Rollback PRs need to have a commit message which contains `[ROLLBACK]`, e.g. "[ROLLBACK] blah blah"
  - Create a GitHub issue for tracking. If and only if this is a manual rollback, label it with the **Rollout Manual Rollback** label and the appropriate rollout repo label
  - Communicate the state of affairs to dncpartners@microsoft.com when the issue is identified along with the tracking GitHub issue, and when it's mitigated.

Hotfixes

- Hotfixes are reactive rollouts done in response to an error in production due to the rollout that just occurred. There are 2 different ways to do this:

    - Make a PR to the Production branch -> rollout to Prod environment -> Make a PR to merge the change back to master.
    - Make a PR to the Staging / master branch -> rollout to staging environment -> test the change -> make a PR to merge the change from master to production branch -> rollout to Prod environment. (This can be done only if staging / master is on lockdown)
    - Manual changes to Prod environment - data changes in the DB, on-prem machine settings etc.
- Workflow for Hotfix:
    - When a service in production is discovered to be in a erroneous state following a rollout.
    - Hotfix PRs need to have a commit message which starts with `[HOTFIX]`, e.g. "[HOTFIX] blah blah"
    - Create a GitHub issue for tracking. If and only if this is a manual hotfix, label it with the **Rollout Manual Hotfix** label and the appropriate rollout repo label
    - If the root cause of failure is determined, make the hotfix and deploy to prod.
    - Communicate the state of affairs to dncpartners@microsoft.com when the issue is identified along with the tracking GitHub issue, and when it's mitigated.
    - The hotfix needs to be communicated to the team and approved by management.

Downtime

- If service downtime occurs as a result of a rollout, an issue must be filed. If alerting files an issue for us, ensure that it is appropriately labeled with **Rollout Downtime** label and the appropriate rollout repo label.
- If the downtime started prior to the issue filing, note the start of the downtime with `Started: [DATE TIME OFFSET]`, not including the square brackets.
- Once the downtime ends, close the issue. If the issue cannot be closed for some reason or the downtime ended prior to the issue closure, the end of the downtime can be noted in the issue body or a comment with `Ended: [DATE TIME OFFSET]`, not including the square brackets.
- If you want to specify the downtime end and start in the same comment or issue body, separate them with a semicolon or a newline, e.g. `Started: [DATE TIME OFFSET]; Ended: [DATE TIME OFFSET]`.
- `[DATE TIME OFFSET]` is formatted as **yyyy-MM-dd hh:mm** [+/-]**hh:mm**, so to represent 1:30 PM Pacific Time on 29 December 2019 you would write **2019-12-29 13:30 -08:00**.

Unit Of Deployment

- All services that are dependent on each other to rollout, are considered a unit of deployment and need to be deployed together in a pipeline.
- Each service in the unit can be in a separate stage for the ease of rerunning a specific service's deployment.
- Deployments to Prod must be approved.
- Any service that is not connected to other services needs to be in a separate pipeline for deployment, in other words is considered a separate unit.

Staging always green

- The staging environment should accurately reflect prod. In other words, if a rollout would break prod it should break staging too.
- Any issues found in staging, regardless of the service, should be communicated to the team (dotnetes@microsoft.com) for transparency.
- Unit/Functional tests will run on staging on every PR Build and block PR merges if tests fail.
- Every service should have post-deployment checks/scenario tests to ensure the services are up and running taking work. If post-deployment checks/tests fail, changes need to be reverted/fixed via PR in Staging to ensure the deployed services in staging are not broken.
- All exploratory / experimental changes need to be done on a dev branch. Any change that is known/expected to temproraily break staging, needs to be communicated to the team (dotnetes@microsoft.com) well in advance and be reasonably timeboxed (case-by-case). This communication should be sent out explaining the period of time in which we expect staging will be down/broken, and put back into a working state by kicking off a build/deploy from master at the appropriate time.
- All merges to staging/master need to stop **two days** prior to the day of rollout for e.g. if the rollout is happening on a Wednesday, the last merge to staging/master should happen on EOD Sunday giving us 2 full days for stabilization. Only changes that unblock the staging build would go into staging/master during the stabilization period.

Was this helpful? 👍 👎