WG LTS BoF - KubeCon NA 2018
Contributor Summit
Dec. 10, 2018

Video link: https://www.youtube.com/watch?v=ZnJe6T18jz8

Below notes are not in chronological order of discussion, rather are binned into topic areas. Also the points recorded aren't necessarily WG LTS or Kubernetes project agreed facts, rather reflect comments/opinions/ideas thrown into this early brainstorming discussion by attendees. They were hand scribed by Tim Pepper, who didn't manage to capture names of speakers, please edit for clarity if you were present.

Discussion started with a couple slides to level set:
https://drive.google.com/open?id=109BitfXVpYPLETbgpoD3S064Rz4FC7q5xzomcFqthIY

(VERY gappy partial) attendee list:

- Aaron Crickenberger, Phil Wittrock, Justin Santa Barbara, Jordan Liggitt, Jago Macleod (Google)
- Bob Wise (Amazon)
- Nick Young (Atlassian)
- Dhawal Bhanushali, Mark Johnson, Tim Pepper, Fabio Rapposelli (VMware)
- Timothy St. Clair (Heptio)
- Hannes Hoerl, Maria Ntalla (Pivotal)
- Davanum Srinivas (Huawei)
- Angus Lees (Bitnami)
- Noah Abrahams (InfoSiftr, CNCF Ambassador)
- Yassine Tijani (OCTO Technology)
- Arnaud Meukam (alter way)
- Mayank Kumar (Salesforce)

Support

- Isn't just having a set of branches that receive patches, but also must have clear upgrade path from prior supported branches to some future supported branch(es)
- Need policy on what patches go into branches
- General agreement community should not allow feature additions to supported branches
- What are the user and cluster operator classes we care about?
    - Some said "all of them", but discussion did include use cases folks do not want to support.
    - There is a multi-modal distribution of requirements and practices. These may or may not be ones the community wants to support, and some splitting of focus between community and vendor/distributor likely makes sense.

Velocity

- Features already take 3+ releases to go from idea to production. This spans at least 9 months and commonly more. Need some level of usage and adoption in the first portions of that time to get feedback on implementation.
- Why don't people use beta releases?
    - Debs/RPMS are missing
    - Package streams aren't available besides stable
    - Distros probably can't fill this gap at the pace of our development. A -devel stream and packages needs to come from the upstream Kubernetes community.
- If we want users to test beta and RC builds, those need quality. Today our beta and RC builds are more calendar based than quality metric based. Quality happens more at the end of the release cycle during code freeze.
- Users and cluster operators want to see longer stable production deployments, less variability.
- Fundamentally disingenuous of us as Kubernetes developers to tell the cloud native story of containerized/ephemeral everything, but then say that's just for apps on top of your cluster and not your cluster control plane. Apps are increasingly agile, but our quarterly cadence feels fairly waterfall by comparison.
- It is possible to move both fast and slow at the same time. It is an established model to have parallel streams for this reason.

Versioning

- What does the "1" in 1.x.y mean? Should it be dropped?
- Unclear criteria for inclusion of features in each 1.x and fixes in each 1.x.y?
- Other projects do "stable" and "devel" streams.
- Concern that "LTS" would be a hard fork that never converges. What is the final state of a branch (ie: how does one get off of the branch)?

Compatibility/Quality

- App developers must have the discipline to only use stable APIs.
- Not enough of the feature set is covered yet by stable APIs.
- Auth and client-go regularly break.
- Normal required APIs must be brought to stable.
    - "Stable" is not just a name, need better testing to prove stability and prevent regression.
- API boundaries
    - Rest API
    - Config API
    - Backend C[R/S/N]I and provider APIs
    - ...these define our boundaries relative to our dependencies and our users.
    - Without these stabilized/tested, we don't have a clear boundary.
- Some folks are shipping full stacks (app + k8s) to insure compatibility (model like shipping a specific JVM with each Java app)

- Quality can't be just at the end of the release cycle during code freeze.
- We need more people fixing bugs and improving test cases.
- We're not ready today for a change in support stance. Community hasn't agreed stability criteria. Our status quo on documented criteria is horrible, but improving. Needs more people working on process and docs.
- We must establish a culture of CI tests always passing green.

Upgrading

- Outage windows: Kubernetes is a different project than most prior ones in computer science. Cloud native architectures more easily allow designing applications for high availability and continually running.
- Do we require users to do an incremental upgrade that visits every release between a prior "stable" or "LTS" branch and the current release?
    - This is the easier, safer path. The current API deprecation policy requires it.
    - Users and cluster operators don't like this requirement if it's visible to them, but perhaps tooling that pipelines a stack of upgrades could be acceptable.
- Skipping releases in upgrade is not safe today.
- Is it possible for tooling to make it safe/easy to upgrade directly across larger delta?
- Compliance auditing and strict version control is important to some enterprises and user types (government, financial, medical, …)
    - What happens to these folks' certifications if they're doing an upgrade from 1.x to 1.(x+5) and it takes a significant amount of time. Are they in violation of something in the meantime? What happens to them if the upgrade fails for some reason and they're in an in between mixed state?
    - If required to only run certified versions, what do they certify?
    - Highly regulated users may not be able to do a pipeline deployment of a stack of upgrades spanning a year's Kubernetes releases.
    - Is there something like a "Windows Service Pack" possible?
- Is it safe to run a cluster with mixed component versions? One attendee is running a mix of 1.10 and 1.12 nodes, because some nodes require the older version still.
- Upgrades have been a thing in tech for a while. Can we at least be as good as normal? Is there pragmatic actions we can take today to improve upgrade to a good enough state, while also looking at longer term ideal?
- Need to collect data from folks who are doing upgrades on their practices and pain points.
    - WG LTS Survey draft: http://bit.ly/k8s-wg-lts-draft-survey
    - SIG Cluster Lifecycle survey: https://bit.ly/2FPfRiZ
- Need a "--dry-run" option for upgrade
- Need solid tooling to migrate data/config across releases (eg: CRD's and CR versioning work underway)

Stable APIs

- Today most anybody using Kubernetes in production is relying on code that is beta and even alpha designated.

Kernel vs Distribution

- If Kubernetes (ie: k/k) is viewed as a OS kernel, what would be the OS userspace around it? What does an integrated Kubernetes kernel+userspace distribution look like?
- As a community we only control Kubernetes and can't declare all associated components' current version as "LTS" in order to enable an integrated distribution that is also "LTS".

Goal: Can we be in a better place at the end of 2019? Need to establish concrete success criteria, such as:

- By Q1 2019 (1.14 release) get KEP process modified to require feature proposals include documentation of upgrade/downgrade path and this be covered in test
- By ?? 2019 strongly define API boundaries
- By end-of-year 2019 bring key APIs to stable
- ...others?