# Tekton Build Cache Konzept

Technisches Konzept für ein Caching System für ESTA Tekton Build Pipelines, zur Optimierung der Buildzeiten von Maven, Npm, Go und Python Builds.

Die grundlegende Evaluation von Caching Technologien wurde im Rahmen eines [Architekturentscheids](file:///C:\pages\viewpage.action%3fpageId=2088675196) gemacht.

* [Grundlagen](#TektonBuildCacheKonzept-Grundlagen)
* [Ablauf Load Cache](#TektonBuildCacheKonzept-AblaufLoadCache)
* [Ablauf Save Cache](#TektonBuildCacheKonzept-AblaufSaveCache)
* [Cache Keys](#TektonBuildCacheKonzept-CacheKeys)
  + [Key Composition](#TektonBuildCacheKonzept-KeyComposition)
  + [Dependency Hash](#TektonBuildCacheKonzept-DependencyHash)
  + [Metadateien](#TektonBuildCacheKonzept-Metadateien)
  + [Recovery Key](#TektonBuildCacheKonzept-RecoveryKey)
* [Access Control](#TektonBuildCacheKonzept-AccessControl)
  + [Variante A)](#TektonBuildCacheKonzept-VarianteA))
  + [Variante B)](#TektonBuildCacheKonzept-VarianteB))
* [Cache Lifecycle](#TektonBuildCacheKonzept-CacheLifecycle)
  + [Periodischer Cache Refresh](#TektonBuildCacheKonzept-PeriodischerCac)
* [Technische Aspekte](#TektonBuildCacheKonzept-TechnischeAspek)
* [Monitoring](#TektonBuildCacheKonzept-Monitoring)
  + [Messwerte](#TektonBuildCacheKonzept-Messwerte)
  + [Auswertung](#TektonBuildCacheKonzept-Auswertung)

## Grundlagen

Das Caching System für Tekton soll ein Optimum an Performance, Disk/Volume Space und Parallelität erreichen.

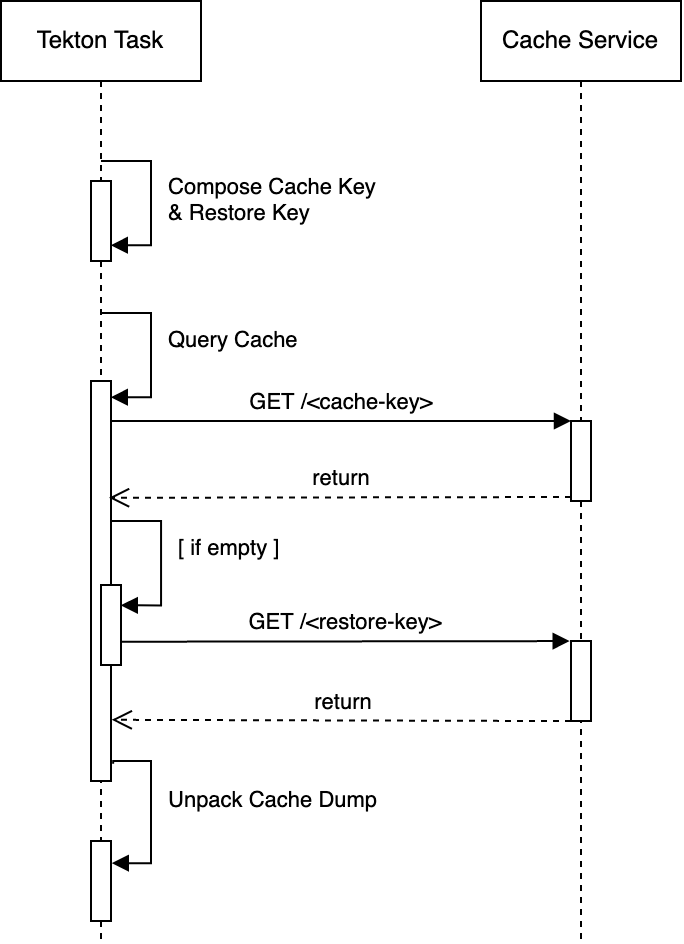
Grundsätzlicher Ansatz für das Caching: **Cache Dumps (Snapshots) in einem S3 Bucket** (→ Variante 4 in [Shared Caches für Tekton Pipelines](https://confluence.sbb.ch/pages/viewpage.action?pageId=2088675196)) gemäss der Idee von [Github Actions](https://github.com/actions/cache).

Das Caching System für ESTA Tekton ist also eine Art Key-Value Store mit Dateien als Values und eindeutigen Keys. Als Backend wird S3 verwendet und für die Verwendung in den Tekton Pipelines werden via Tekton Builder Images zwei Funktionen (Binaries) angeboten: "**Load Cache** from Store" und "**Save Cache** into Store". Zusätzliche Hilfsfunktionen z.B. für die Generierung von Cache-Keys können ebenfalls via Builder Image angeboten werden.

## Ablauf ****Load Cache****

Wird vor dem eigentlichen Build Tasks ausgeführt, um einen bestehenden Cache Dump aus dem Store zu laden und ins lokale Dateisystem des Pods zu entpacken.

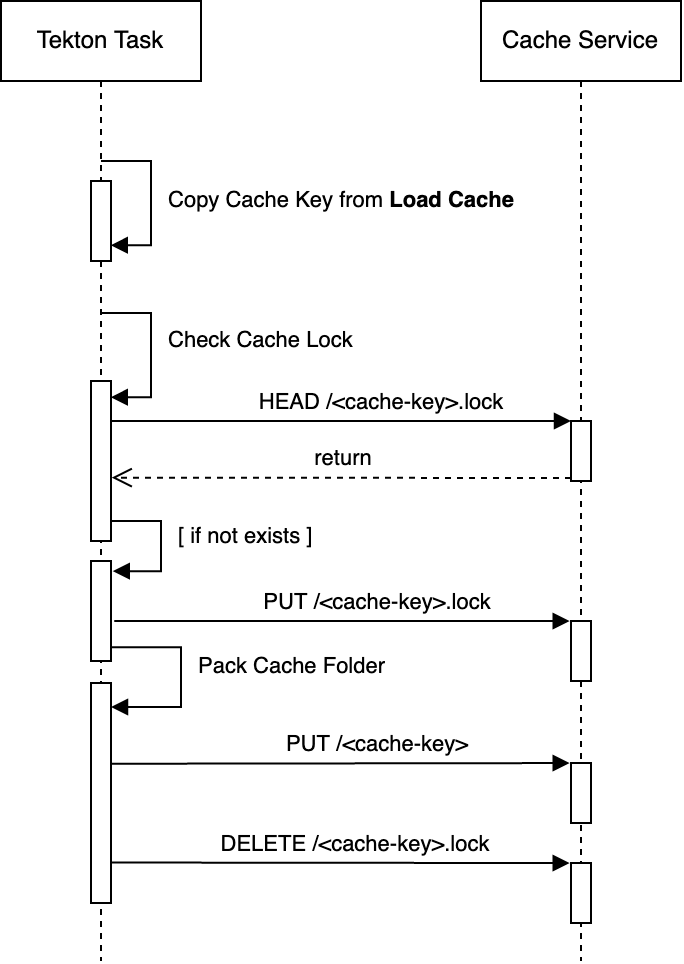
1. Hash berechnen basierend auf Lock Files (e.g. package-lock.json)
2. Cache-Key mit Hash erstellen
3. Abfrage von Cache-Entry mit diesem Key
4. Fallback auf neuste hash-loses Entry (restore-key)
5. Download des Cache-Dumps und Entpacken in Pod Volume



## Ablauf ****Save Cache****

Wird nach dem Build Task ausgeführt, um den lokalen Cache als kompletten Dump persistent im Store zu speichern.

1. Hash berechnen basierend auf Lock Files
2. Cache-Key mit Hash erstellen
3. Bestehendes Cache-Entry mit diesem Key prüfen
4. Cache-Lock mit dem berechneten Key prüfen
5. Wenn kein Entry und kein Lock existieren:
   1. Cache-Lock auf diesen Key registrieren (mit TTL ca. 5 Minuten)
   2. Lokalen Cache komprimieren und hochladen
   3. Cache-Lock entfernen
6. Falls Entry oder Lock existiert: Skip



## Cache Keys

Cache Keys sollen den Zustand einer bestimmten Dependency-Konstellation für ein Projekt (Repository) repräsentieren. Davon ausgehend, dass als Storage Backend S3 verwendet wird, beinhalten die Cache Keys auch / um eine hierarchische Dateistruktur auf dem Cache Service aufzubauen. Einzelne Ordner können auf S3 mit IAM Policies (z.B. Access Restrictions) oder Lifecycle Rules belegt werden.

**Beispiel:** clew-tekton/kd-esta--esta-metrics/maven-v1-2022.2-qfhewoiffhqawsoifhqaf

**Pattern:** <tekton-namespace>/<repo-slug>/<package-manager>-<cache-api-version>[-<date-slot>]-<hash>

### Key Composition

* tekton-namespace: Name des Tekton Namespace. Access Policies werden auf dieser Ebene enforced.
* repo-slug: Unique Key des Projekt Repositories (Tekton Label tekton.esta.sbb/repository)
* package-manager: Package Manager/Technologie (maven, npm, go, poetry, etc.)
* cache-api-version: Version des Cache Service APIs
* date-slot: Zeitliche Komponente des Cache Eintrags (optional)
* hash: Dependency Hash

### Dependency Hash

Der Hash repräsentiert die aktuelle Liste aller Projekt-Dependencies mit (exakten) Versionen. Er soll als String von 24 Zeichen basierend auf Metadateien des spezifischen Package-Managers berechnet werden. Für die verschiedenen Technologien resp. Package-Manager (Maven, Npm, Go, Poetry) müssen also unterschiedliche Metadateien verwendet werden. Die Berechnung des Hashes soll basieren auf der Checksumme aller Metadateien erfolgen.

**Beispiel NPM:**

for file in `find . -name "package-lock.json" -not -path "./node\_modules/\*"`; do shasum $file >> checksums.txt; done

shasum checksums.txt | head -c 24

### Metadateien

* Maven: pom.xml
* NPM: package-lock.json (Fallback auf package.json)
* Go: go.sum (Fallback auf go.mod)
* Python: poetry.lock (Fallback auf pyproject.toml)

### Recovery Key

Bei ändernden Dependencies ergibt sich ein neuer Hash und damit wird kein Cache-Eintrag mit dem aktuellen Key gefunden. Um nicht mit einem leeren Cache zu starten, wir der letzte Cache-Eintrag für den Cache Key ohne den Hash als Ausgangszustand geladen. Der Recovery Key entspricht also dem Cache Key ohne den Dependency Hash. Zum Beispiel: clew-tekton/kd-esta--esta-metrics/maven-v1-2022.2-

Das Lookup des Recovery Keys erfolgt über ein Listing alles Keys, welche mit dem Recovery Key beginnen, sortiert nach Erstellungsdatum.

## Access Control

Ausgangslage: Für die Speicherung wird ein globaler S3 Bucket für alle Caches aufgesetzt.

Anforderung: Tekton Namespaces dürfen nur auf "eigene" Cache Dumps zugreifen (read/write) können.

Die Separation nach Namespaces erfolgt durch Unterordner im S3 Bucket.  
Für das Enforcement von Access Policies auf diesen Namespace-Ordnern stehen aktuell zwei Varianten zur weiteren Prüfung:

* A) Individuelle Access Tokens für S3 + IAM Roles pro Namespace
* B) Cache-Access Proxy in Kubernetes/Openshift prüft Herkunft (Namespace) von Anfragen

Grundsätzlich soll eine Create-only (write-once-read-many - WORM) Policy auf dem S3 Bucket eingerichtet werden. Damit wird verhindert, dass Cache Dumps (gewollt oder ungewollt) verändert werden können. Dies kann mittels [Object Lock](https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lock.html) oder indirekt über [Versioning](https://stackoverflow.com/questions/10592541/amazon-s3-acl-for-read-only-and-write-once-access/10592928#10592928) in S3 umgesetzt werden.

### Variante A)

Mittels [IAM Policies](https://aws.amazon.com/premiumsupport/knowledge-center/s3-folder-user-access/) kann der Zugriff auf Ordner und Unterordner für jeden Tekton Namespace definiert werden. Voraussetzung dafür sind jedoch individuelle IAM Access Tokens für jeden Tekton Namespace. Diese, sowie die zugehörigen IAM Policies müssen beim Setup eines Tekton Namespace angelegt werden.

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| Native Steuerung und Konfiguration direkt in S3 und IAM | Registrierung von IAM Roles, Policies und Access Credentials beim Setup notwendig |
| Enforcement und Monitoring mit AWS Tools | Access Credentials im Tekton Namespace vorhanden |

Beim Setup eines Tekton Nameaspace müssen IAM Roles, Policies und Access Credentials auf AWS registriert werden und als Secret (Access Credentials) gespeichert werden. Wie das automatisiert gemacht werden soll (Terraform, ~~Crossplane~~, andere) und was der Aufwand dafür ist, muss noch evaluiert werden.

### ~~Variante B)~~

In dieser Variante verfügt der Tekton Namespace über keinen direkten Zugriff auf den S3 Bucket (kein Access Token). Sämtliche Requests werden an einen Proxy Service auf demselben Cluster geleitet, welcher mit Network Policies auf OpenShift gesteuert wird.

Dieser Proxy Service prüft die Herkunft des HTTP Requests anhand von HTTP Headers und identifiziert somit den Tekton Namespace als "Requester" und stellt sicher, dass dieser nur auf Bucket Ressourcen in seinem Cache Ordner zugreifen darf.

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| Setup von Tekton Namespace unverändert | Neue Komponente von ESTA entwickelt, deployed und maintained |
| S3 Access Token nur in dediziertem Namespace vorhanden | Monitoring muss selbst aufgesetzt werden |
|  | Machbarkeit noch offen |

Voraussetzung für Variante B) ist, dass die Herkunft eines HTTP Request innerhalb von Openshift (aus welchem Pod/Namespace) sicher geprüft werden kann. Das ist ohne zusätzliche Tools (z.B. ServiceMesh) oder Hacks (ServiceAccount Token mitschicken) nicht möglich. Der Implementationsaufwand für diese Variante wird gemäss ersten Untersuchungen vermutlich **unverhältnismässig hoch** ausfallen.

## Cache Lifecycle

Bei der Verwendung von S3 als Storage Backend kann mittels [Lifecycle Configuration](https://docs.aws.amazon.com/AmazonS3/latest/userguide/how-to-set-lifecycle-configuration-intro.html) eine automatische Löschung von "alten" Cache Einträgen eingestellt werden. Damit entfällt ein pro-aktives periodisches Prüfen und Aufräumen des Caches.

Als Vorschlag soll beim Aufsetzen des S3 Buckets eine max. Vorhaltezeit von 7 Tagen auf Cache Dateien konfiguriert werden.

Für .lock Dateien soll max age auf 5 Minuten eingestellt werden.

### Periodischer Cache Refresh

Durch die Verwendung der date-slot Komponente im Cache Key kann eine periodische (z.B. monatlich oder quartalsweise) Erneuerung der Caches erreicht werden.

TODO

Vor- und Nachteile von diesem Feature müsse noch evaluiert werden.  
[ESTA-5237](https://flow.sbb.ch/browse/ESTA-5237) - Abrufen der Vorgangsdetails... STATUS

## Technische Aspekte

* Für die Kompression von Cache Dumps soll Tar + [Zstandard](https://github.com/facebook/zstd) verwendet werden.
* Für das Entpacken der Caches im Tekton Task (Pod) soll ein spezifisches Volume verwendet werden.  
  → zu prüfen ist noch, ob ein emptyDir ausreichend ist (Achtung: [Ephemeral Storage Limit](https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#local-ephemeral-storage) auf Openshift Nodes) oder ob dafür pro Pipelinerun ein Workspace PVC angelegt werden muss.

## Monitoring

Die Cache Verwendung (Cache Hits/Misses), die Cache Grösse sowie die Zeit für Load und Save Operationen als auch die Build Performance insgesamt soll periodisch überprüft und monitored werden)

TODO

Ein Konzept zum Monitoring wird separat ausgearbeitet (siehe unten).  
[ESTA-5238](https://flow.sbb.ch/browse/ESTA-5238) - Abrufen der Vorgangsdetails... STATUS

### Messwerte

|  |  |
| --- | --- |
| **Messwert** | **Metriken/Quelle** |
| Cache Hits | Als Tekton Result speichern/auslesen [~~S3 Request Metrics~~](https://docs.aws.amazon.com/AmazonS3/latest/userguide/metrics-dimensions.html#s3-request-cloudwatch-metrics) ~~(Get requests)~~ |
| Cache Misses | Als Tekton Result speichern/auslesen [~~S3 Request Metrics~~](https://docs.aws.amazon.com/AmazonS3/latest/userguide/metrics-dimensions.html#s3-request-cloudwatch-metrics) ~~(4xx errors)~~ |
| Cache Grösse | [Amazon S3 dimensions](https://docs.aws.amazon.com/AmazonS3/latest/userguide/metrics-dimensions.html#s3-cloudwatch-dimensions) / [S3 Console oder S3 Storage Lens](https://aws.amazon.com/blogs/storage/find-out-the-size-of-your-amazon-s3-buckets/) |
| Load Times | [Tekton Metrics](https://tekton.dev/docs/pipelines/metrics/) (TaskRun Duration; \*task=esta-cache-load) |
| Save Times | [Tekton Metrics](https://tekton.dev/docs/pipelines/metrics/) (TaskRun Duration; \*task=esta-cache-save) |
| Overall Build Performance | [Tekton Metrics](https://tekton.dev/docs/pipelines/metrics/) (PipelineRun Duration) |

### Auswertung

Die Messwerte sollen mit Prometheus gesammelt und in einem [Grafana Dashboard](https://grafana-clew-esta-dev.apps.aws01t.sbb-aws-test.net) visualisiert werden. Die Tekton Metriken sind bereits verfügbar, für die S3 Metriken muss Cloudwatch eingerichtet und ein [Prometheus Fetcher](https://promcat.io/apps/aws-s3) aufgesetzt werden.

Falls die Integration der S3 Metriken in Grafana zu aufwändig wird, können die Metriken auch über die AWS Console (Bucket metrics) angeschaut werden.