

Flink Discrepancy Analysis

Team Debeggars

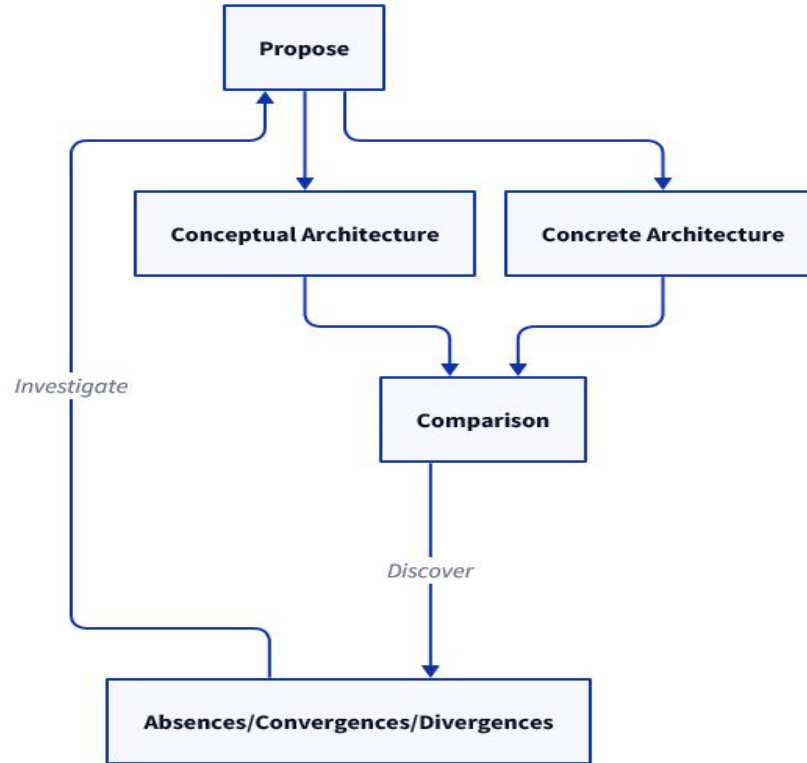


Overview

- Process of analysis (Investigation approaches)
- Concrete vs Conceptual
- Divergences analysis
- Evolution of JobManager
- Use Case Implications
- Overall Implications
- Limitations of findings
- Learned Lessons

Overall Process

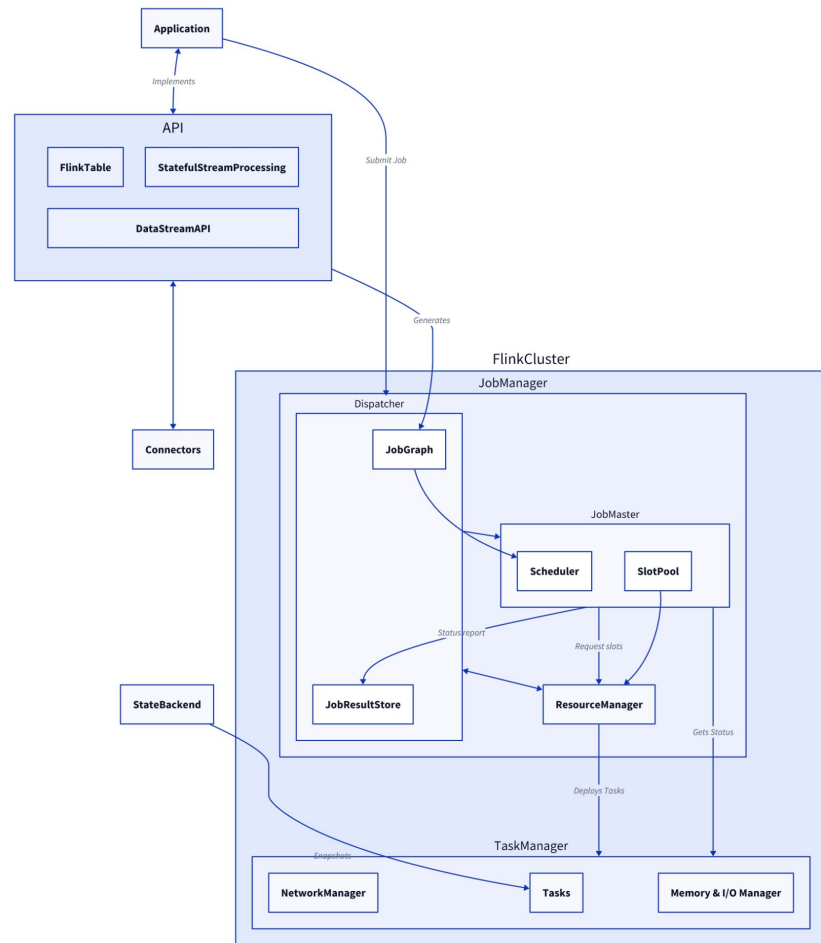
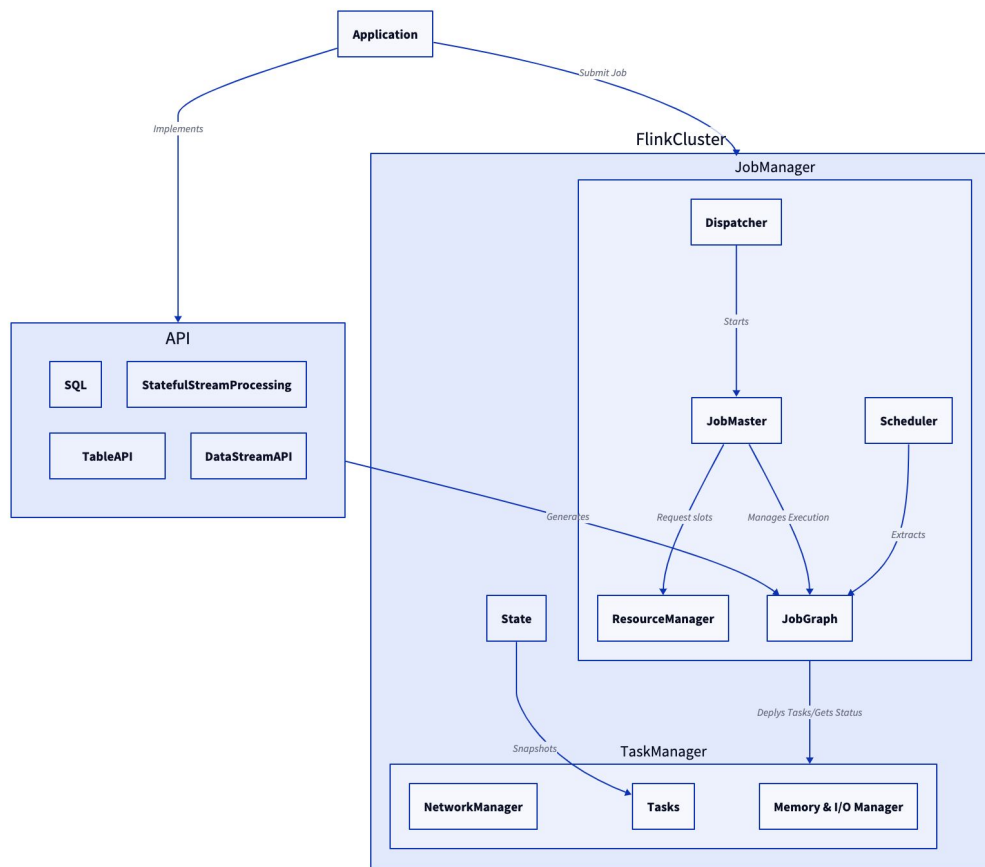
- Propose
- Compare
- Investigate

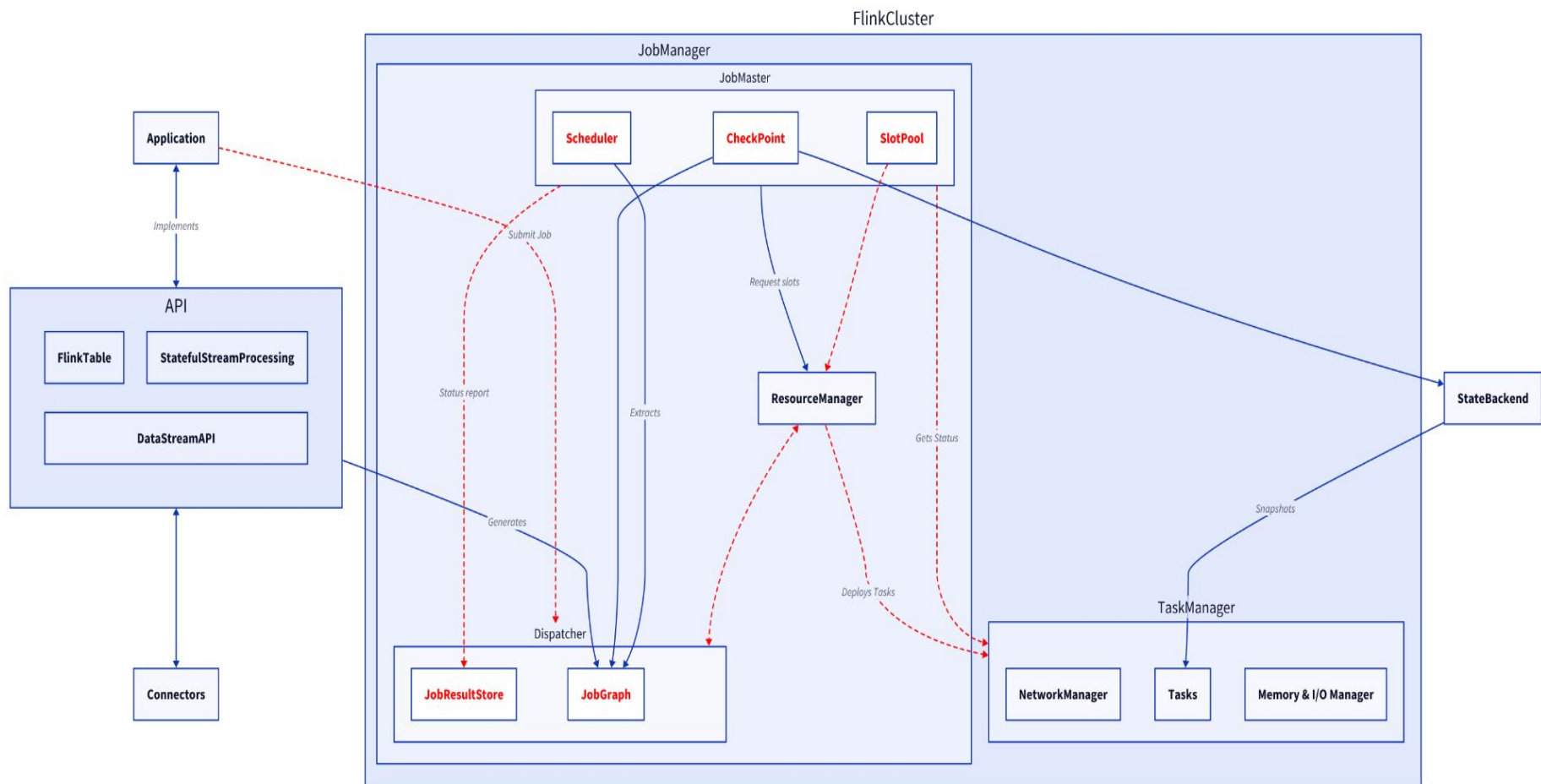


Investigation approaches

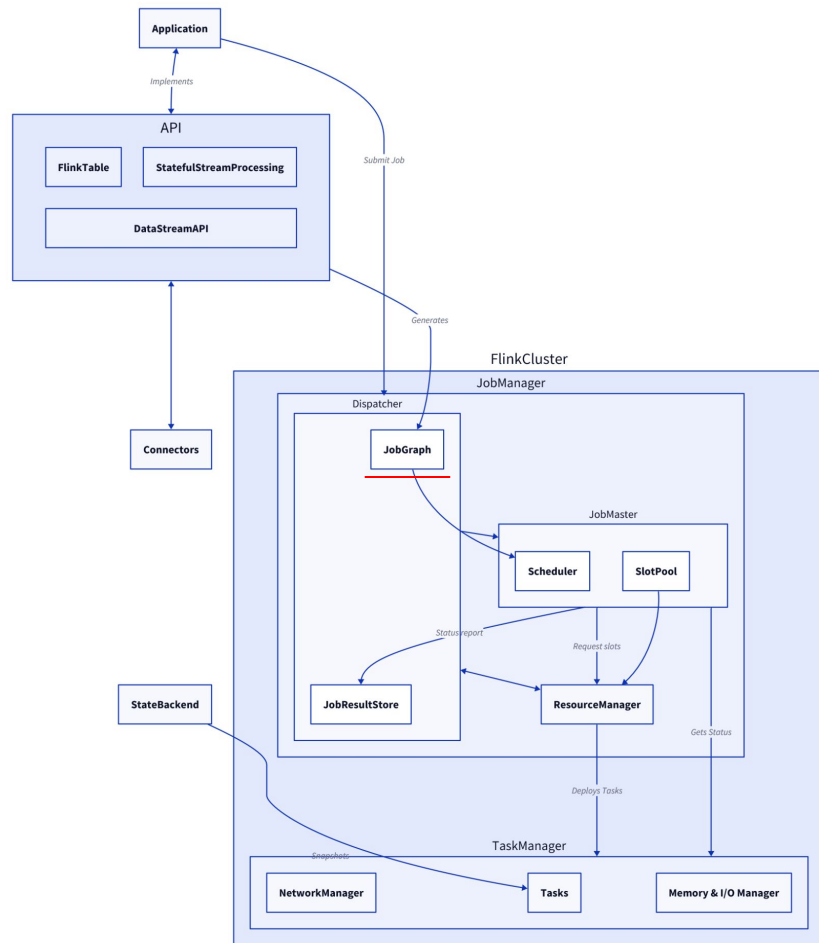
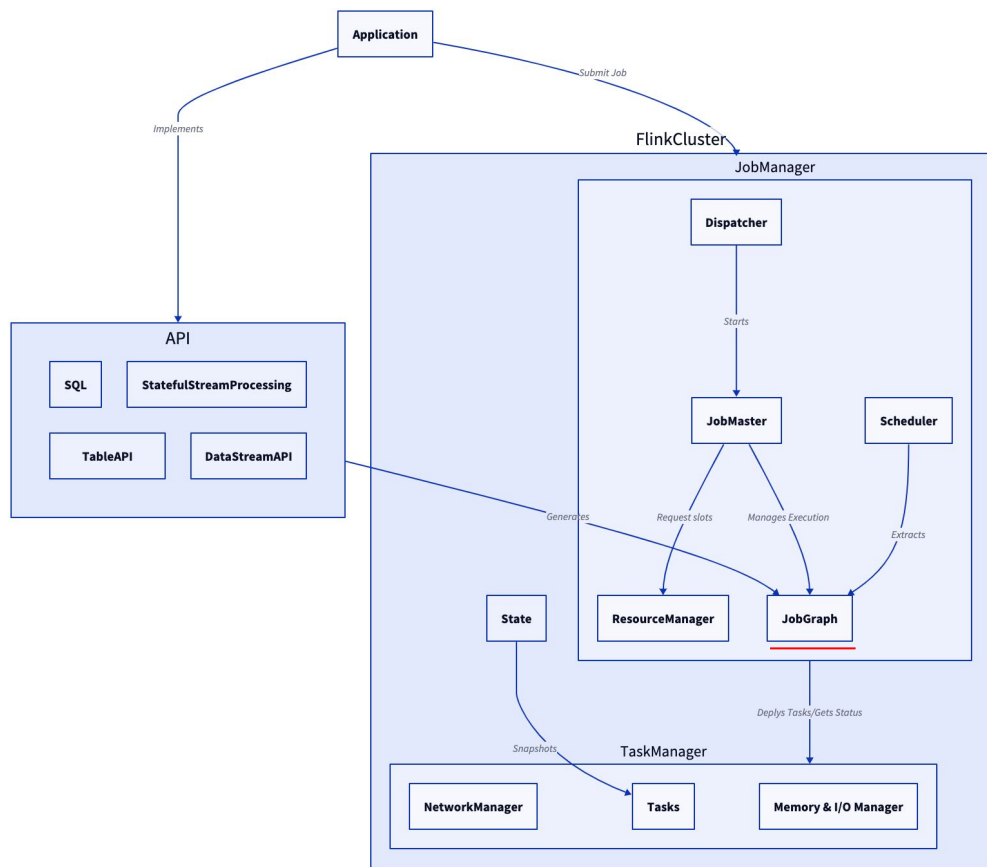
- Apache Flink Confluence to search for rationale.
- Apache Flink 1.17.1 JavaDoc.
- Apache Flink source code, GitHub Blame.
- Trace PR's and JIRA tickets.
- Expert blog posts, answers on Stack Overflow.

Conceptual vs Concrete





JobGraph



The JobGraph subsystem

[FLINK-11843] Allow to pass JobGraphStore into Dispatcher via Dispatc...

...herServices

This commit introduces DispatcherFactoryServices which is an extension of PartialDispatcherFactoryServices. The extension allows to set a specific JobGraphStore which is being forwarded via the DispatcherServices to the Dispatcher.

Implementation of a Dispatcher leader service that oversees the dispatchers and corresponding JobGraphs.

Fix issue of loss of persistence as JobGraph was loosely coupled with Dispatchers.

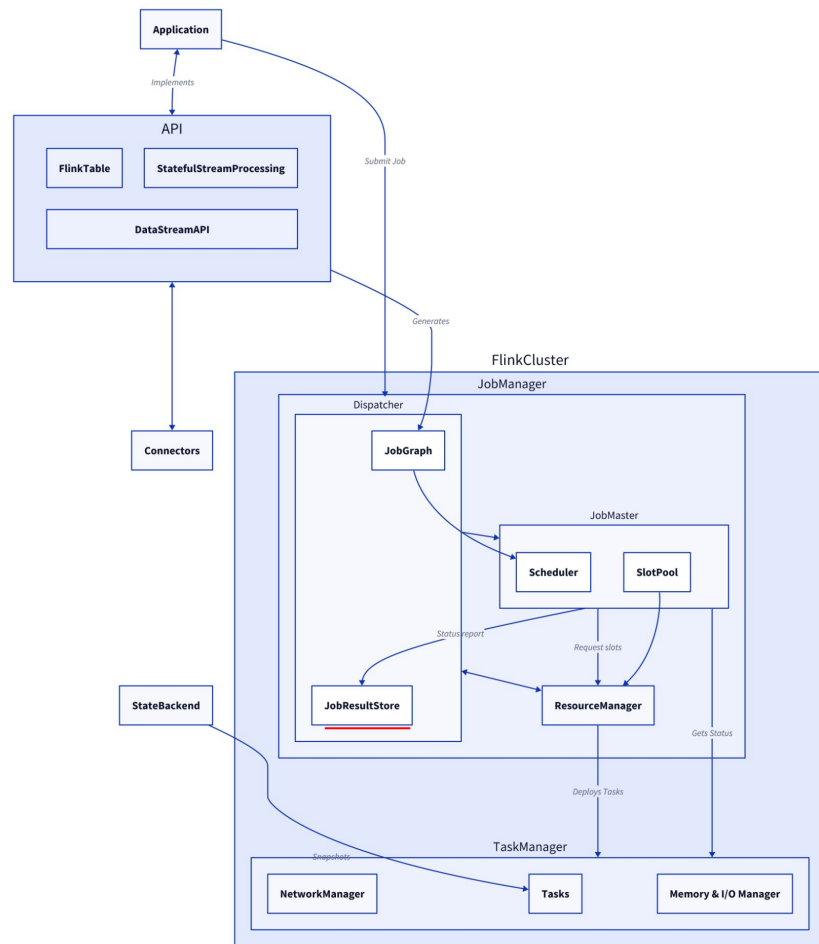
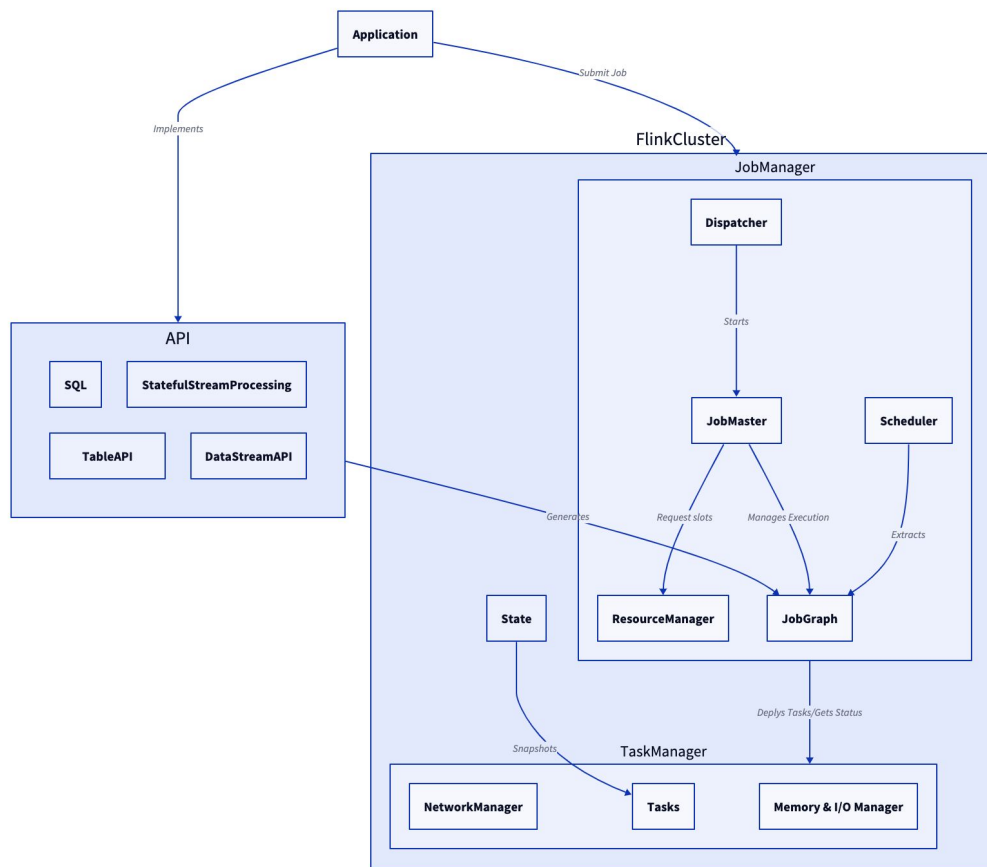
Bug Fix/ Improvement for the purpose of high availability.

<https://issues.apache.org/jira/browse/FLINK-11843>

By: Till Rohrmann

1.10.0 2019

JobResultStore



JobResultStore



Flink / FLINK-11813 Standby per job mode Dispatchers don't know job's JobSchedulingStatus / FLINK-25430

Introduce JobResultStore

Details

Type:  Sub-task

Status:

RESOLVED

Priority:  Major

Resolution:

Fixed

Affects Version/s: None

Fix Version/s:

1.15.0

Component/s: None

Labels: [pull-request-available](#)

Description

This issue includes introducing the interface and coming up with a in-memory implementation of it that should be integrated into the Dispatcher.

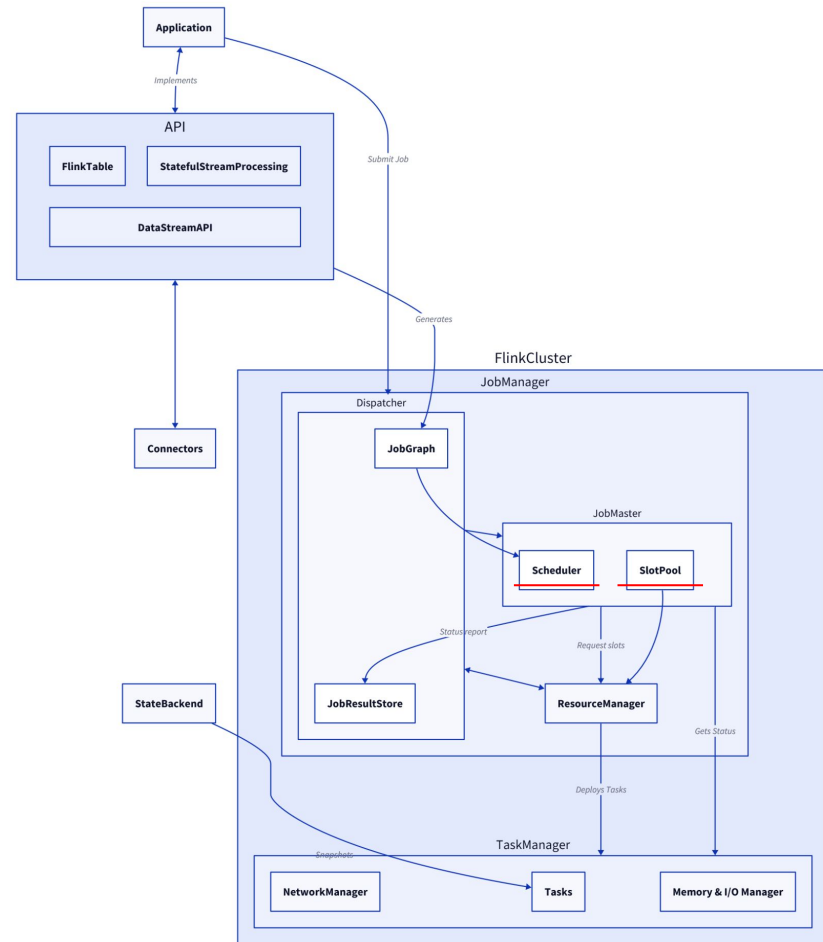
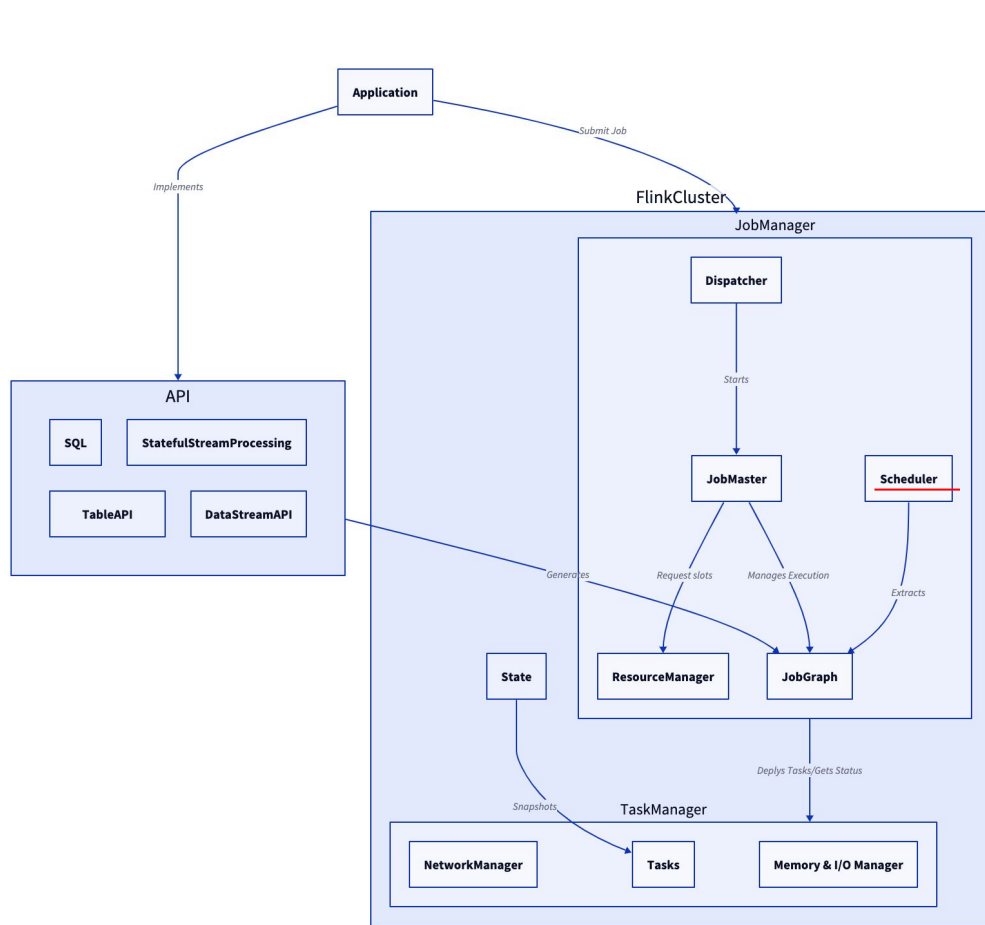
- We'll introduce the new interface `JobResultStore`
- We'll remove the `RunningJobsRegistry` a replace its functionality with `(Standalone|Embedded)JobResultStore` (This is basically only about `RunningJobsRegistry#setJobFinished` method)
- The `JobResultStore` should be initialized along the `JobGraphWriter` since both components are closely related.

By: Matthias Pohl

1.15.0, 2022.

<https://issues.apache.org/jira/browse/FLINK-25430>

Scheduler/ SlotPool





Scheduler/ SlotPool




Flink / FLINK-10429

Redesign Flink Scheduling, introducing dedicated Scheduler component

Details

Type:  New Feature
Priority:  Major
Affects Version/s: 1.7.0
Component/s: Runtime / Coordination
Labels: None

Status:  CLOSED
Resolution: Done
Fix Version/s: None

Description

This epic tracks the redesign of scheduling in Flink. Scheduling is currently a concern that is scattered across different components, mainly the ExecutionGraph/Execution and the SlotPool. Scheduling also happens only on the granularity of individual tasks, which make holistic scheduling strategies hard to implement. In this epic we aim to introduce a dedicated Scheduler component that can support use-case like auto-scaling, local-recovery, and resource optimized batch.

The design for this feature is developed here: <https://docs.google.com/document/d/1q7NOqt05HIN-PIKEPB36JiuU1lu9fnxxVGJzylhsxU/edit?usp=sharing>

<https://issues.apache.org/jira/browse/FLINK-10429>

1.7.0



Extract scheduling-related code from SlotPool

Details

Type:	Sub-task	Status:	CLOSED
Priority:	Major	Resolution:	Implemented
Affects Version/s:	1.7.0	Fix Version/s:	None
Component/s:	Runtime / Coordination		
Labels:	pull-request-available		

Description

The other half of the current scheduling logic is the management of slot sharing and is located in the SlotPool. We need to extract this logic into our new Scheduler component from the previous step. This leaves us with a simpler SlotPool that mainly cares about obtaining, holding, and releasing slots in interaction with a ResourceManager. The new Scheduler can now identify slot sharing groups and interacts with the SlotPool.

Issue Links

contains

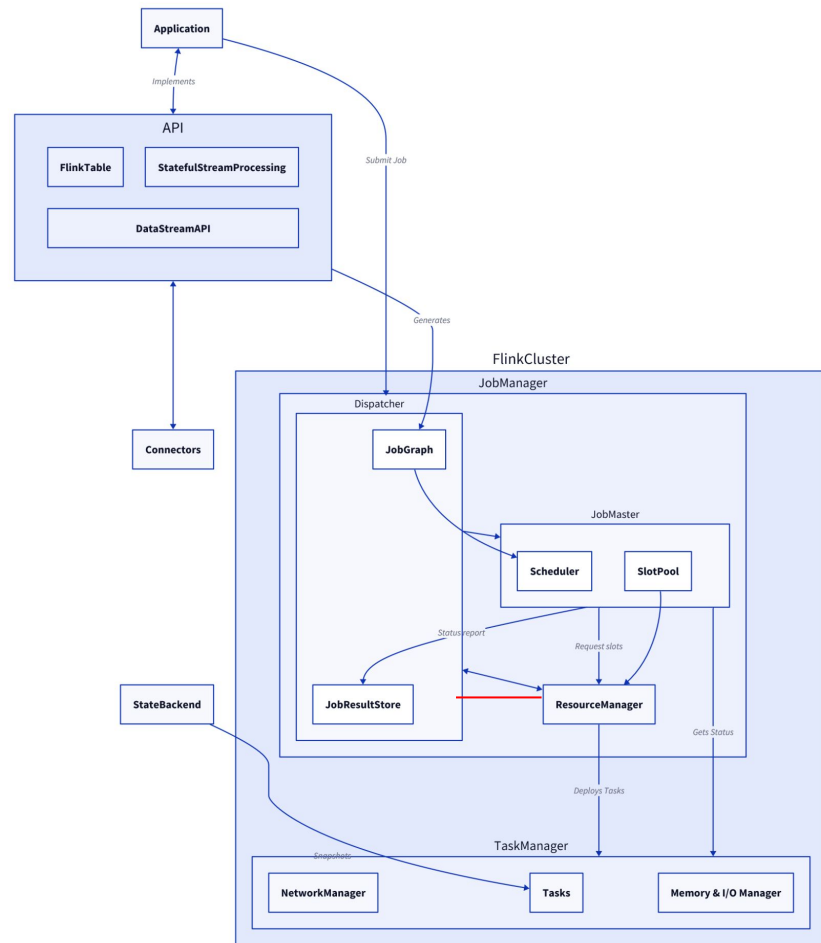
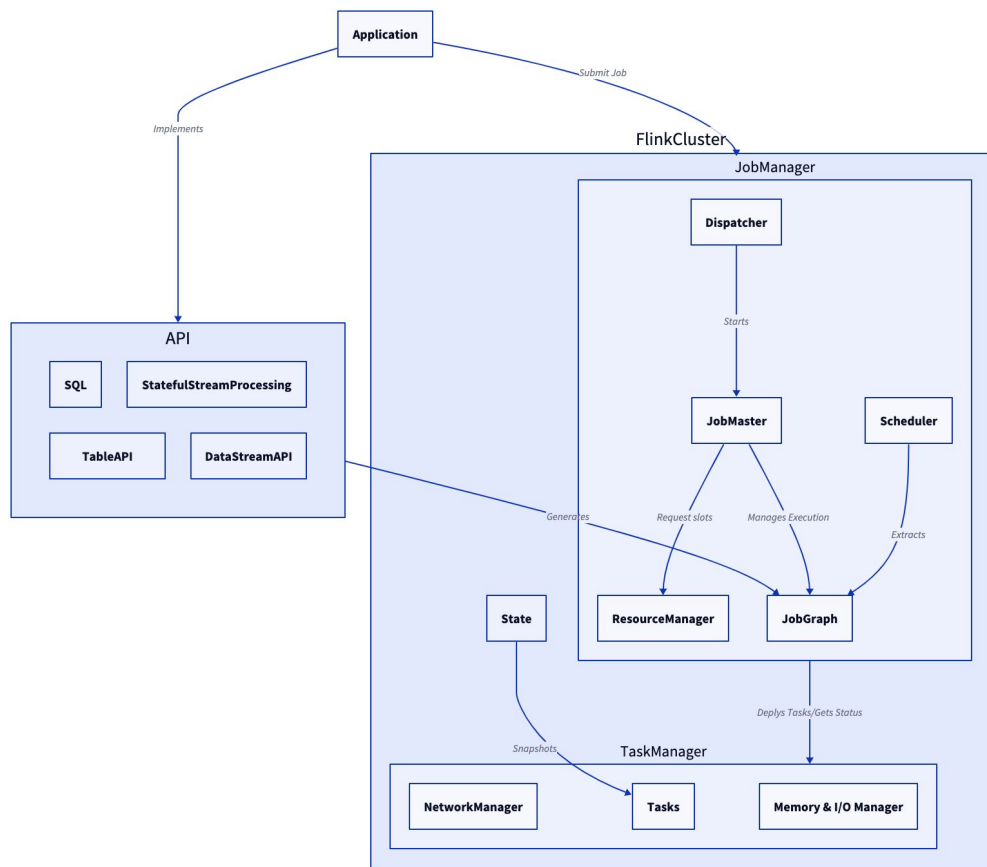
[FLINK-11375](#) Concurrent modification to slot pool due to SlotSharingManager releaseSlot directly RESOLVED

links to

[GitHub Pull Request #6898](#)


[GitHub Pull Request #7662](#)

ResourceManager and Dispatcher interdependency



ResourceManager and Dispatcher interdependency



[FLINK-10411] Introduce DispatcherResourceManagerComponentFactory ...

tillrohrmann committed on Sep 28, 2018

org.apache.flink.runtime.entrypoint.component

Verified

9f5fd07



Class DispatcherResourceManagerComponent

java.lang.Object
org.apache.flink.runtime.entrypoint.component.DispatcherResourceManagerComponent

All implemented interfaces:
AutoCloseable, AutoCloseableAsync

```
public class DispatcherResourceManagerComponent
extends Object
implements AutoCloseableAsync
```

Component which starts a Dispatcher, ResourceManager and WebMonitorEndpoint in the same process.

 Flink / FLINK-10411
Make ClusterEntrypoint more modular

Details

Type:Improvement

Priority:Minor

Affects Version/s:1.7.0

Component/s:Runtime / Coordination

Labels:pull-request-available

Status:CLOSED

Resolution:Fixed

Fix Version/s:1.7.0

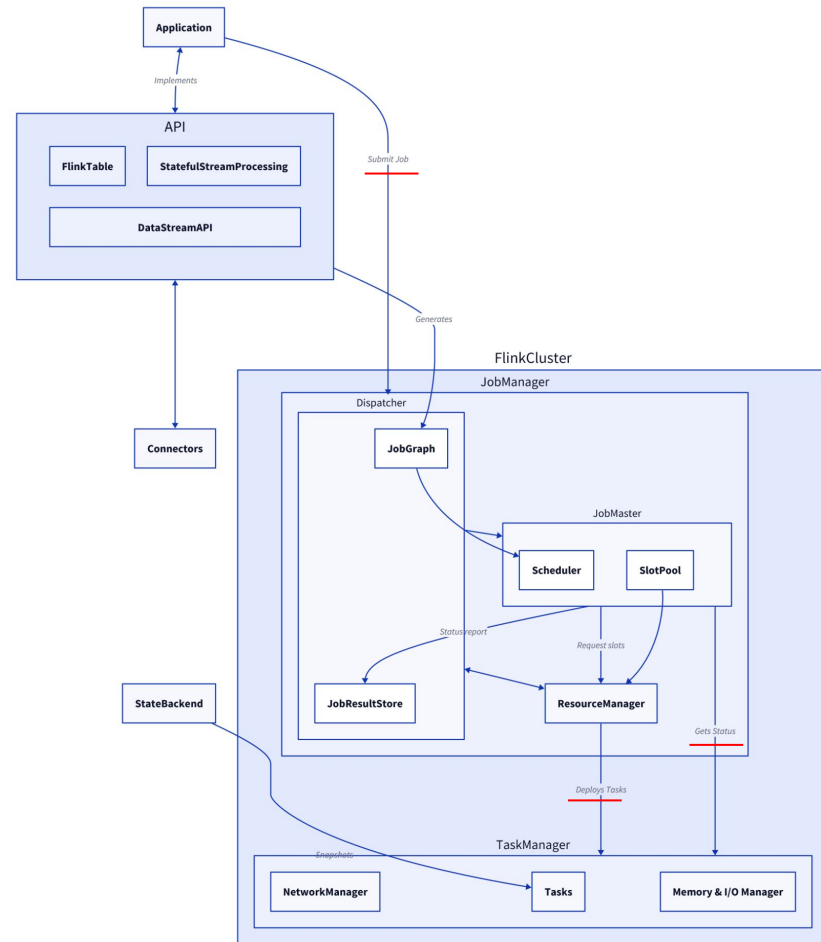
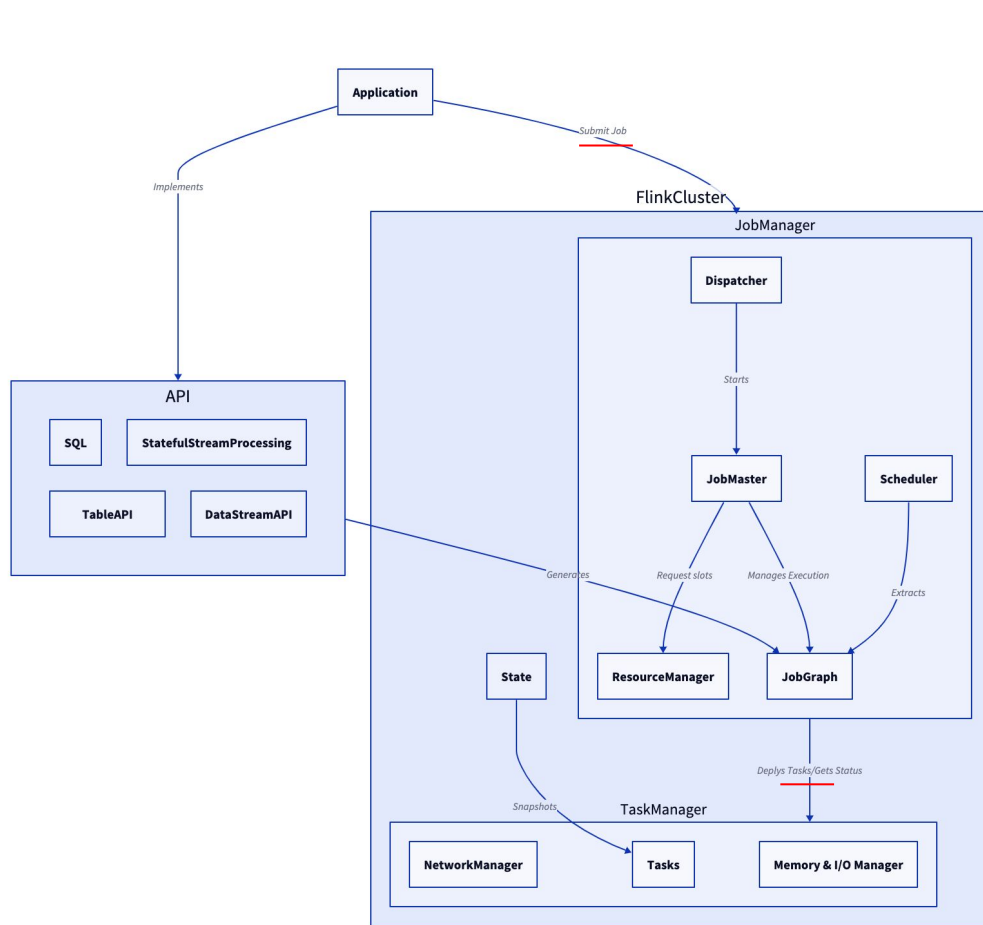
Description

Currently, the ClusterEntrypoint is not very modular in the sense that it cannot be really used for testing purposes (e.g. starting a Dispatcher with a WebMonitorRestEndpoint). The problem is that the ClusterEntrypoint combines too many responsibilities (creating the cluster services, starting the cluster components and deciding on when to terminate the JVM process).

I suggest to make the structure more compositional, meaning to split up the service generation from the cluster component start up. That way we could also remove code duplication between the different ClusterEntrypoint implementations.

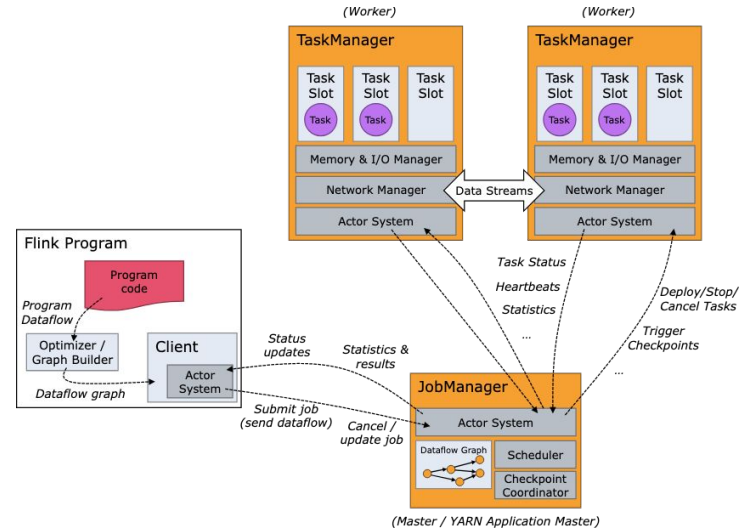
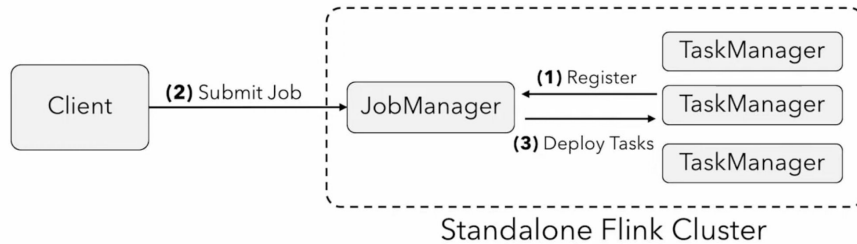
To implement Compositional start up of different components as an improvement for testability, code duplication, performance etc.

Does not couple ResourceManager and Dispatcher too significantly at all.



The Evolution of the Flink JobManager - V1.0

- One large JobManager.scala file
- No concept of ResourceManager or Dispatcher
- Handled everything - Job lifecycle, scheduling, TaskManagers



The Evolution of the Flink JobManager - FLIP 6

- Flink Improvement Proposal 6 drastically changed the JobManager:
 - JobManager handles only a single job
 - ResourceManager lives across jobs and JobManagers
 - ResourceManager must be able to fail without interfering with the execution of current jobs
 - TaskManagers are both in contact with the ResourceManager and JobManager
 - The dispatcher accepts job submissions from clients and starts the jobs.
- Instead of a single JobManager, we now have a JobManager, ResourceManager, and Dispatcher.



The Evolution of the Flink JobManager - Flink 1.9

- “The Flink Master is the master of a Flink Cluster. It contains three distinct components: Flink Resource Manager, Flink Dispatcher and one Flink JobManager per running Flink Job” - Flink 1.9 glossary
- JobManager.scala file removed (in 1.8 release)



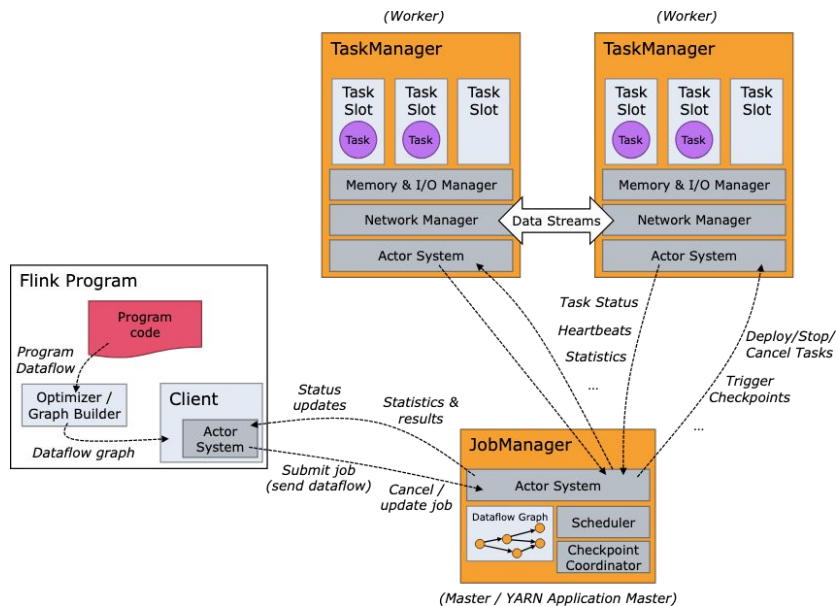
The Evolution of the Flink JobManager - Flink 1.11+

- “The JobManager is the orchestrator of a Flink Cluster. It contains three distinct components: Flink Resource Manager, Flink Dispatcher and one Flink JobMaster per running Flink Job” - Flink 1.11 glossary

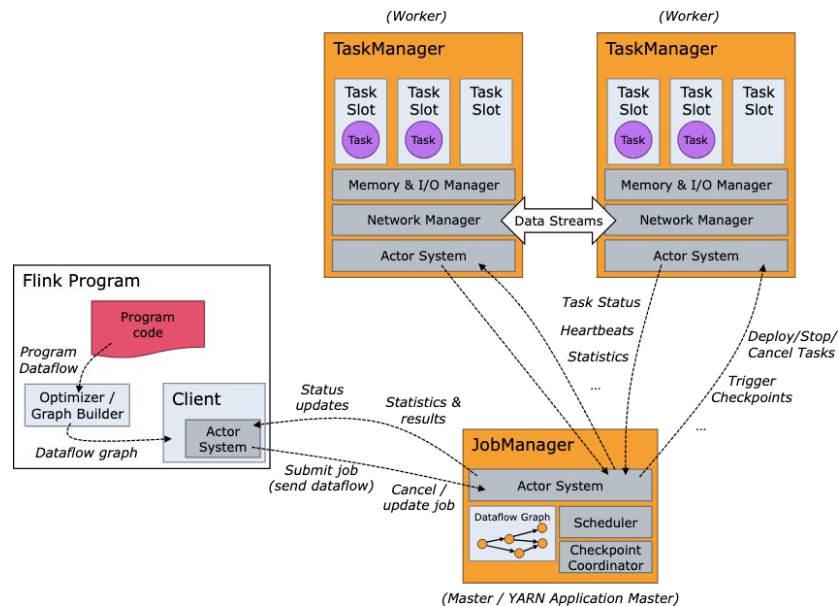


According to the docs...

V1.0 Architecture Diagram



V1.17 Architecture Diagram



JobManager vs JobMaster Confusion

```
private JobManagerRunner createJobMasterRunner(JobGraph jobGraph) throws Exception {
    Preconditions.checkNotNull(jobManagerRunnerRegistry, "jobManagerRunnerRegistry");
    return jobManagerRunnerFactory.createJobManagerRunner(
        jobGraph,
        configuration,
        getRpcService(),
        highAvailabilityServices,
        heartbeatServices,
        jobManagerSharedServices,
        new DefaultJobManagerJobMetricGroupFactory(jobManagerMetricGroup),
        fatalErrorHandler,
        failureEnrichers,
        System.currentTimeMillis());
}
```

???

Confused with JobManager and JobMaster

Asked 3 years, 2 months ago Modified 2 years, 8 months ago Viewed 360 times



I am new to Flink.

1

On the internet, I have always seen the concept `JobManager`, but when I look into the Flink source code(the latest code forked from `master` branch)



Interface: `JobManagerRunner`

javadoc: Interface for a runner which executes a `{@link JobMaster}`.



class: `JobMaster`

`JobMaster` implementation. The job master is responsible for the execution of a si

I would ask whether `JobMaster` in the code is exactly the concept of `JobManager` on the internet

[apache-flink](#)

Share Edit Follow



Get updates
on questions
and answers



asked Aug 31, 2020 at 11:32

Tom

5,922 12 44 107

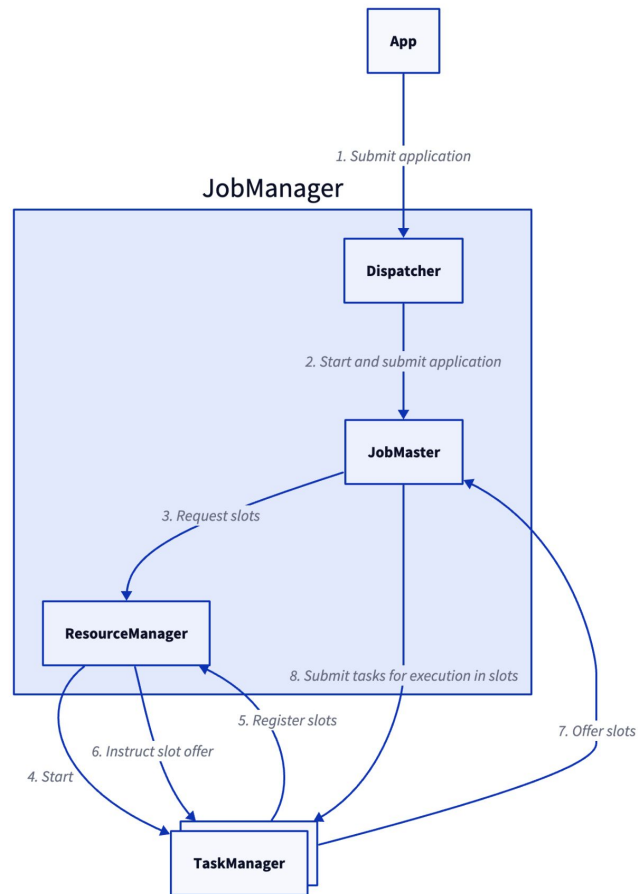
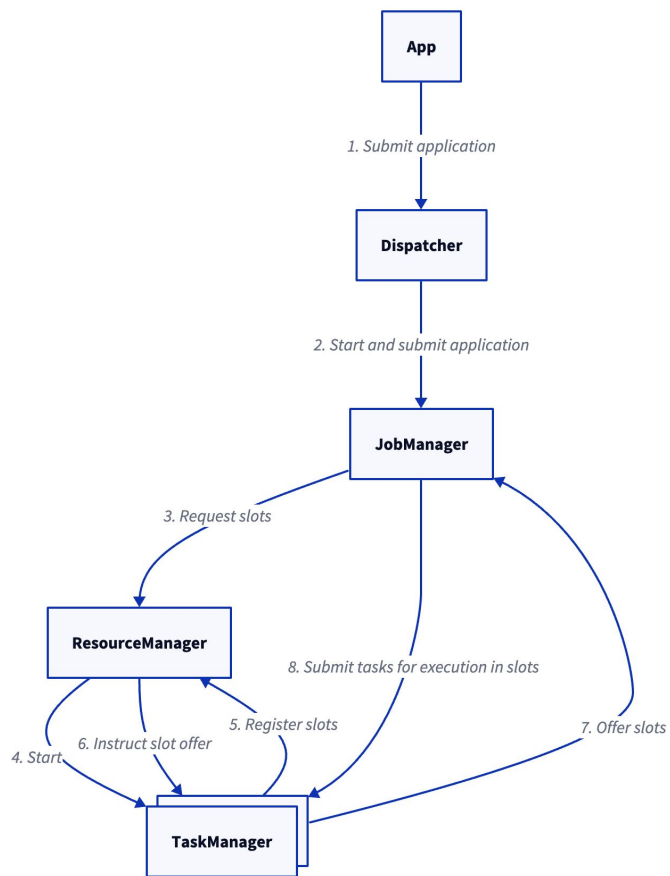
1 Answer

Sorted by: Highest score (default)

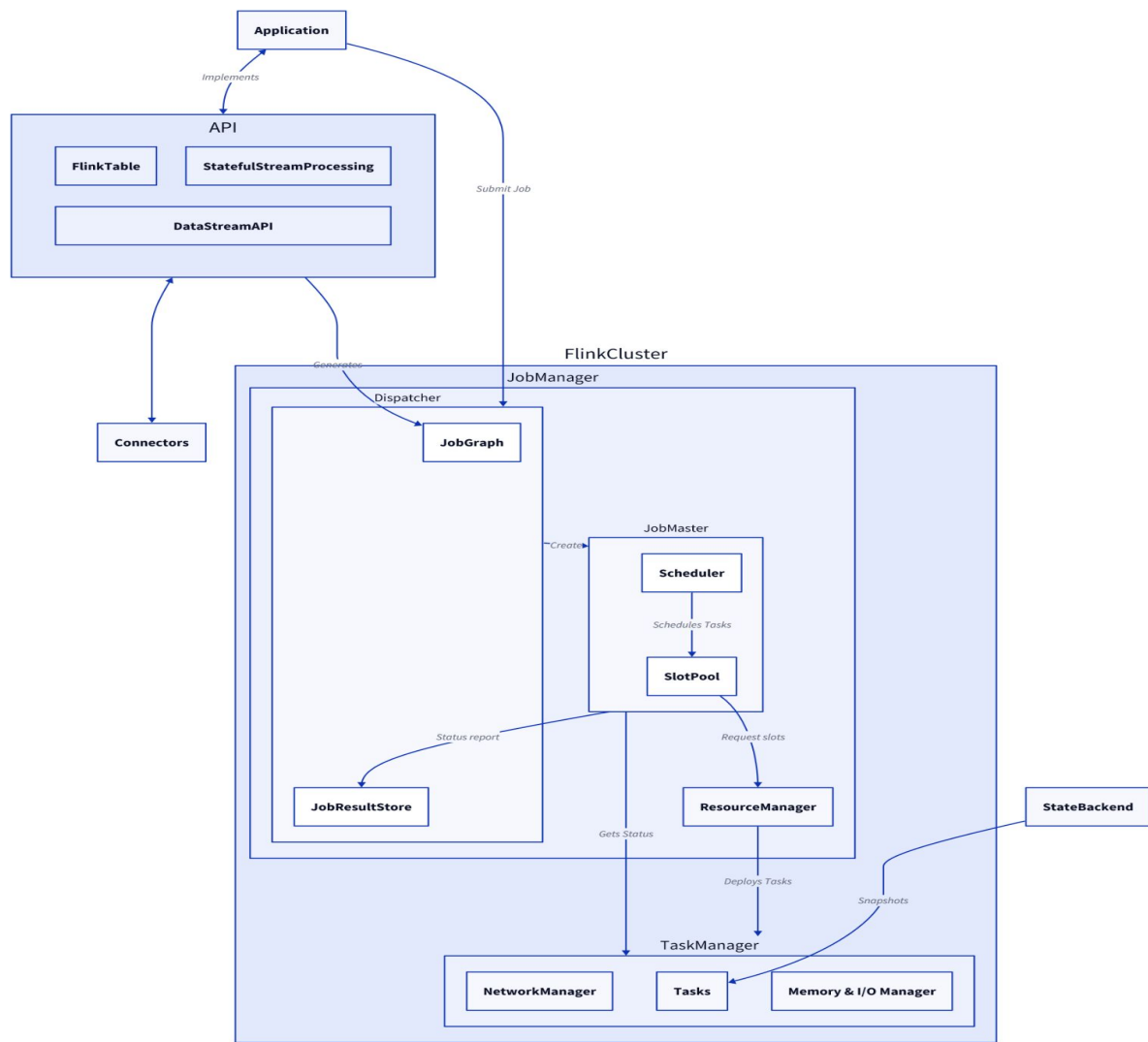


3

Short answer: the `JobManager` referred to in the docs comprises the `Dispatcher`, a cluster-framework-specific `ResourceManager`, the `Blob Server`, and a set of `JobMasters` (one per job). You won't see a `JobManager` class in the code; while there was once a monolithic `JobManager`, it was refactored by FLIP-6 into a set of separate components.



Revised Architecture



Use Case Implications

- No implications to the use case scenarios
- Implications to how each use case interacts with the Flink components
 - With regards to the JobMaster/JobManager

Use Case 1

Scaling large amounts of data for an E-commerce site for a personalized shopping experience

- JobMaster analyzes data instead of the JobManager
 - JobManager is a collection of the three components

Use Case 2

Online payment fraud detection

- Dispatcher submits an application to the JobMaster instead of the JobManager
- JobMaster performs the previously mentioned roles of the JobManager
 - JobManager is made up of the JobMaster, ResourceManager, and Dispatcher

Implications of these differences

- The overall JobManager component of Flink maintains same responsibilities
 - The end user doesn't really need to be concerned with it.
- Complex/Confusing Codebase:
 - Greater barrier of entry for new developers
 - current developers will have trouble maintaining existing code
- Release notes and development notes do not provide full picture
 - Quality of comments and text entered by developers in the past

<https://ieeexplore.ieee.org/document/1311060>

Limitations of findings

- Quality of the Confluence/JIRA descriptions and comments - Lack of context and unclear rationale.
- Constant changes to the code (Improvement process, new feature, bug fixes) - Requiring additional human judgement to discern rationale, and the effect the changes have on the architecture.
- Apache Flink is a big project - limited scope of analysis.

Learned lessons

- **Lesson 1:** The rationale behind differences between Conceptual architecture and concrete architecture must be understood clearly from multiple perspectives (4 W's). The rationale dictates the scale of impact on the architectural design of the software.
- **Lesson 2:** Difference analysis, investigation and creating new conceptual/concrete architecture diagrams should be a process that is iterated multiple times, especially for large software projects, in order to present clearer depiction of the software architecture.
- **Lesson 3:** For large software projects, especially ones that are open-source. It may be incredibly difficult, if not completely impossible, to adhere to a strict conceptual architecture. Hence, it is important to keep documentations accessible and up-to-date.
- **Lesson 4:** Understanding the general direction of software development is a great first step to difference analysis. Confluence helped us a lot.

Conclusion

- Explored Apache Flink's JobManager evolution from a single JobManager.scala file from version 1.0 to later versions, highlighting ResourceManager and Dispatcher components introduction.
- Addressed the challenges in under both conceptual and concrete architectures of Apache Flink.
- Emphasized the importance of clear documentation.

Key lessons iterated:

- Understanding architectural changes from multiple perspectives.
- The iterative process in analyzing and updating architecture diagrams.
- Challenges in maintaining a strict conceptual architecture in open-source environments.
- Importance of up-to-date documentation and awareness of software development trends.



Thank you!