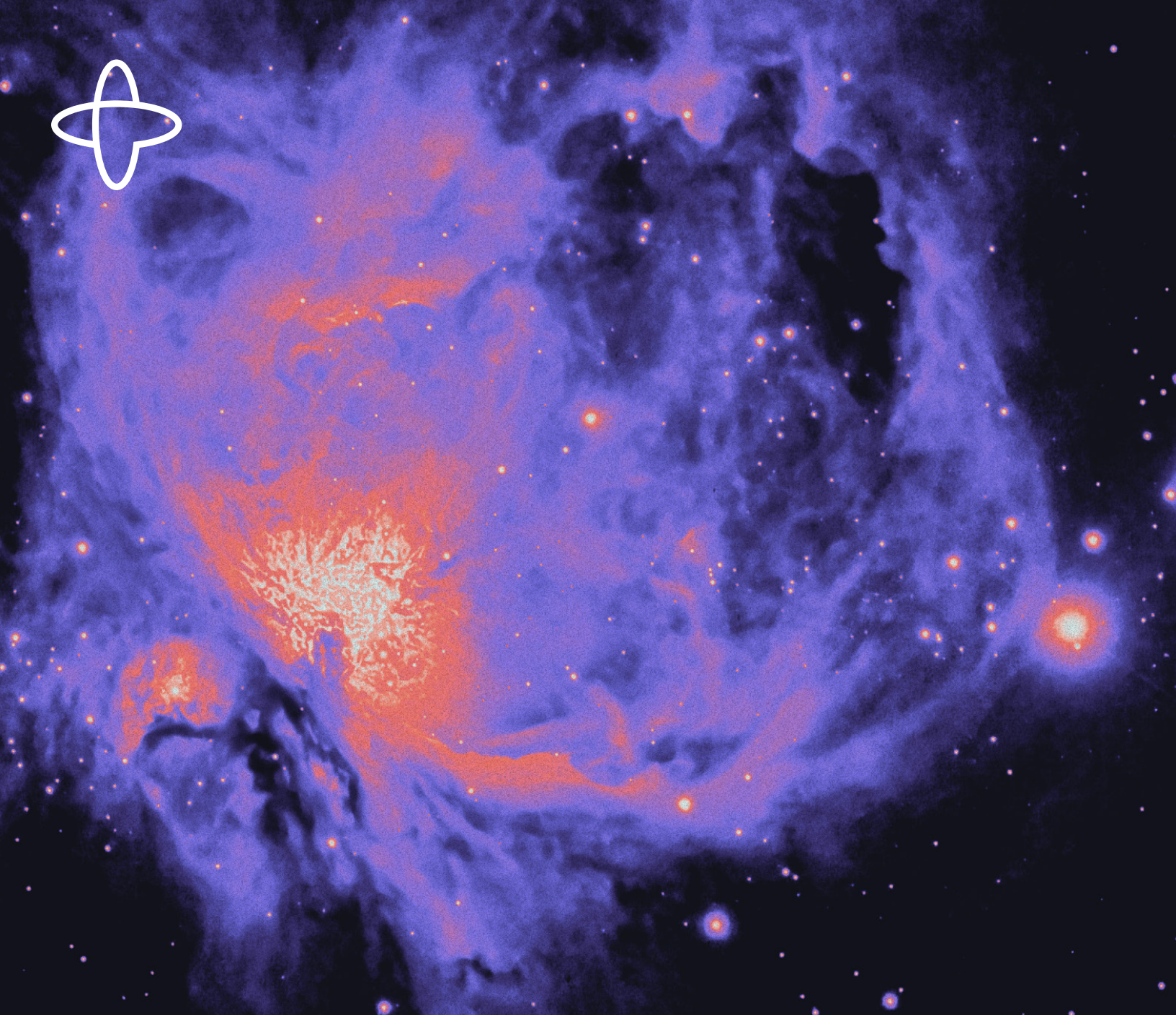


TECH BRIEF

Why Temporal

v.1.0

An Introduction to Temporal

Coding for more reliable systems

To build more reliable systems, we often employ a system architecture that strings together queuing systems and hardware redundancies; however, everything fails. Queues can get overloaded. Database and API calls can timeout. Adjacent services may be unreliable. Networks are not always stable. Furthermore, with modern systems design, we're often tasked with sharing state across a web of services. For the developer, these complexities can become overwhelming, as they need to code for all the things that can go wrong within (and beyond) their system architecture.

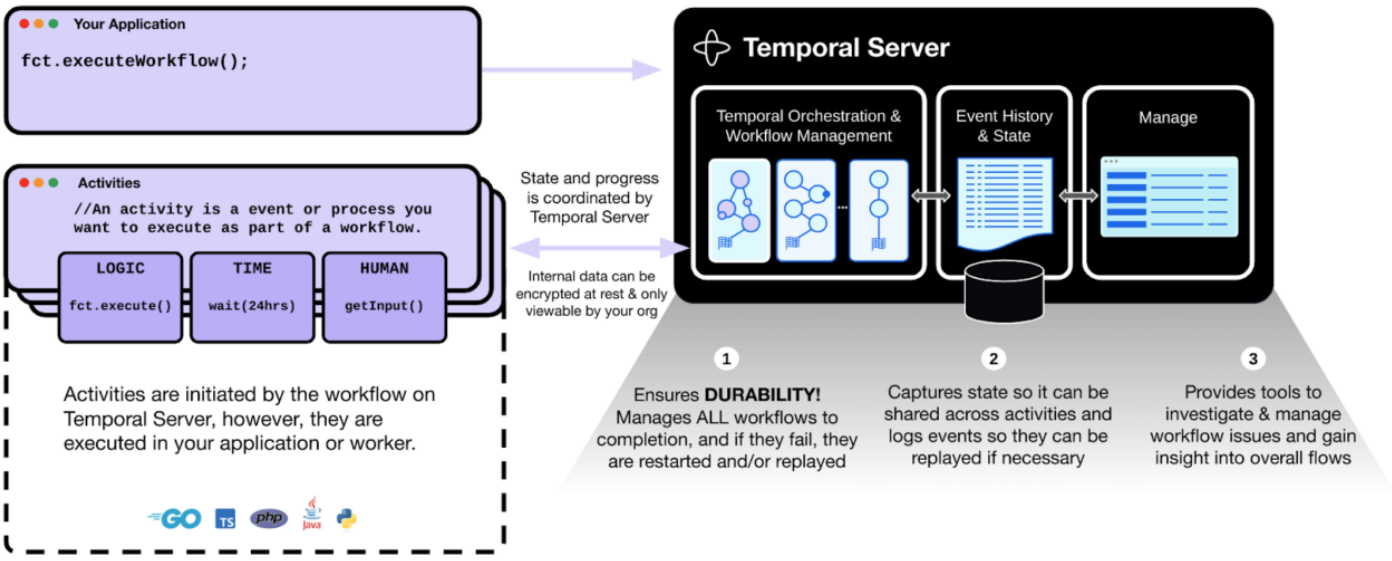
```
Temporal is an open source programming model that can simplify your code, make your applications more reliable, and help you deliver more features faster.
```

It abstracts away the complexities around retries, rollbacks, queues, state machines, and timers so that developers can focus on business logic and not the complexities of failure. With Temporal, you can codify workflows and processes to guarantee consistency for cross-service transactions and durable execution of explicit tasks over any length of time.

The founders of Temporal have iterated over this model a few times in the past fifteen years as tech leads at AWS, Microsoft and Uber. They have created a platform that is both developer friendly while also being extremely valuable to an architect.

Temporal and Durable Workflows

A Temporal Workflow allows you to model and manage both simple tasks or a string of complex procedures so that they are executed reliably to completion, or they will elegantly fail. Developers can define a workflow in the language of their choice and then deploy it and its supporting activities so that Temporal can coordinate the execution of those steps, ensuring that they are executed in the correct order and that all dependencies are satisfied. And if there is a failure, a retry or rollback can be executed.



How does Temporal work?

Temporal consists of two main components, an SDK and the Temporal Server.

You code your application in whatever language you desire using one of the SDKs, which are available for Go, Java, PHP, Python, Typescript, and .NET. The SDK allows you to define a Temporal Workflow and its subsequent Activities that you want to execute reliably. The Workflow runs as a service that long polls the Temporal Server against a task queue for work. These Workflows can be started on the Temporal Server using the SDK and you can set retry policies for them so that they remain durable.

The Temporal Server orchestrates workflows, maintaining workflow state while also providing visibility into any explicit execution of your defined series of activities. The Temporal Server ensures this by placing workflow and activity tasks on an internal queue so they can be picked up and executed in your environment. Temporal provides a strong guarantee to progress workflows to completion regardless of any errors encountered along the way.

Your code. Your data. Your environment.

Temporal employs a unique execution Inversion paradigm. It requires no data from you to operate, nor any of your code. Your code is run on your infrastructure and it uses workers and activity queues to coordinate with a Temporal Server (self hosted or our managed service). Additionally, all data is encrypted both at rest and in motion. And with the Temporal Data Converter, you can provide end-to-end protection as Temporal will merely pass a pointer to your data on your infrastructure and your key, managed by you, is used to decrypt.

Temporal in *your* code

Modern developer tooling and distributed systems have allowed us to build and deliver applications, faster and better than ever. However, we are yet to modernize the stack above the infrastructure layer and challenges for the software developer remain. There are few key issues which Temporal addresses:

Distributed, idempotent and atomic transactions

Idempotency is a guarantee that no matter how many times you execute an explicit transaction, you achieve the same result. And atomic transactions (all or nothing commits) seem easy, but what about transactions that span multiple services? Without Temporal, you have to wrapper the process, manage state and manually check progress against a transaction.

Rollbacks and retries

A process or transaction consists of a few or many steps. It could be as simple as a write to a database or a complex flow of components in a global supply chain. Without Temporal, developers are left modeling failure scenarios and coding complex retry logic to compensate for a deficiency in execution. And with distributed services, ongoing management and coordination of this logic spirals out of control.

Cross service state management

As we breakdown functions into services, we need to share state across linked or even disconnected, polyglot services. Many turn to a database to handle this, but introduces latency, cost and risk to the architecture. Temporal Workflows give your distributed applications an autosave for state that can be shared and even prompt new functions to be initiated

Timeouts and long running processes

Any and every API (database, system, human, etc) will fail and developers are left to deal with logic to define how to wait and what to do when the API ultimately fails. And how do you deal with a process that needs to wait for an elongated amount of time (day, week, month)?

These are just a few areas where Temporal helps developers deal with complexity and become more efficient in the delivery of code. This offloads difficult tasks, allows them to focus on business logic and ultimately deliver more reliable software.

Temporal in use

Companies of all sizes have used Temporal to create and deliver more durable applications. Large engineering organizations like Snap, Netflix, Datadog, Yum Brands, Sofi and Twilio as well as emerging companies like Turo, Descript, and Checkr have all adopted Temporal. Ultimately, they use Temporal as a general purpose, developer platform across a fairly wide variety of use cases, such as:

Transactions: Temporal can wrapper transactions to ensure they compete fully or elegantly fail. Typical transaction data might include: Money, Financial assets, Inventory, Orders

Business Processes: Temporal Workflows model an end to end process and allow you to replay existing events or fail elegantly. Typical business processes might include: ordering food or goods, building a car, background checks and more.

Entity Lifecycle: Temporal allows for a long running process to execute on time without external dependencies and within your code. Typical entity lifecycle processes might include: monthly statements, Inventory management, menu/ listing versioning, etc.

Operational Process: Temporal is often used to automate mundane, high volume, repeated, operational tasks. Typical operational processes might include: provisioning Saas, management of compute resources, automating the software supply chain and moving data between systems.

A Commitment to Open Source

Temporal is licensed using the MIT license, which is one of the most permissive OSI certified licenses available. All Temporal project code is freely available to use, inspect and extend, and is available in a github repository. In order to run Temporal, you'll need to operate the Temporal Server, which typically requires some Kubernetes expertise and you will also need to attach a database as the persistence layer.

Temporal Cloud - All the value as a fundamentally better service

As noted, in order to run Temporal, you must deploy and manage a fairly complex tech stack. Temporal Cloud offloads this activity into a consumption based service so it scales as you scale and you only pay for the value you get from the service. Temporal Cloud also employs a unique control plane that will accommodate spikey workloads without the need of overprovisioning your instance, and it can scale well beyond a self-hosted instance. It is substantially more cost effective, reliable and scalable than managing Temporal yourself.

Temporal Cloud also includes the necessary support and services needed to onboard you successfully, deploy and design your application efficiently and scale your instance as you need. Our team are the architects and makers of the project and we have the most broad set of experience to help you get successful with any project.

Temporal: Move fast and elegantly fail

The use of Temporal represents a paradigm shift for developers and once they see and use it, they can't "unsee" it. While the learning curve can be steep, the benefits of the programming model are enormous as you can greatly simplify your code and ultimately deliver more reliable features, faster. Finally, it is open source and polyglot so you can start using it today in your language of choice.

For more information, please visit **temporal.io** or contact us at **team@temporal.io.**