State Management and Containers

BY JUSTIN WARREN

STATE MANAGEMENT AND CONTAINERS

The stateless nature of containers has proven extremely useful. Rather than dealing with complex maintenance of the operating environment--patching the OS, library versions, and so on--the application hosted in a container can assume its environment is as fresh and clean as when it was first installed, largely because it was only created a few moments ago. Each container deployment is like a shiny new server without the weirdness that grows on a constantly maintained system that has been online for years.

However, this Eternal Sunshine of the Spotless Mind approach has serious limitations. All of the interesting things about applications are data related. While the application you're using to read this text might be stateless, without the state--the text itself--you would have nothing to read, and the application would be somewhat useless.

So it is with all applications, be they banking apps (state tells you how much money you have), video apps (Netflix is pointless without movies) or messaging apps (an empty Facebook is deathly boring). Without state, the applications have no point.

We clearly need some way to manage state, but how?

Requirements for Stateful Solutions

Storing data locally is the first option that most would consider, and it has some advantages over a remote service. The primary advantage is latency; data that is closer to the CPU takes less time to access. For any application that interacts with users, latency can have a major impact on the user experience. Sitting waiting for a webpage to update after you submit a form is tedious, and for more interactive experiences, the latency is even more important.

However, node-local storage is only useful for short-lived persistence that doesn't need to be shared with other nodes. The local storage becomes inaccessible if the node fails, so for true persistence, the data needs to be stored outside the node somehow, and copied to more than one node to provide resilience against failures.

This implies some sort of clustering service, which gets us back into the territory of a storage service. Building a state replication service yourself is a fool's errand, and you're much better off using something created by other smart people.

For a production service, at least two locations are required for resilience, which adds new data replication issues relating to distance and the latency of synchronization. Modern data offerings also provide a range of data services such as snapshots, deduplication, and efficient change replication that solve many of these issues. We must also be mindful of the need for performance. The storage layer needs to have both predictable low-latency, and sufficient bandwidth for the volume of data being processed.

The access mechanism to this storage service depends on the kind of data you need to store. Highly structured data, or data that has specialized indexing and retrieval needs, is best suited to storage in a database of some kind, but for all other data, a shared filesystem is a good choice.

Containers Are About Process

A major reason for using containers is the software development process it encourages. Modern organizations are adopting an industrialized approach to developing software, as compared to the more artisanal handicrafts approach of yesteryear. This industrial approach is characterized by its rapid iterations and mass-produced copies of essentially identical components using assembly line style processes.

Containers are destroyed and rebuilt from scratch because this is faster and easier than constantly mutating a long-lived system. Configuration drift and other minor differences between environments cause quality and reliability issues. Ensuring that the baseline image is the same everywhere helps to reduce variance. The idea isn't new, and borrows heavily from the Toyota Production System and Lean Manufacturing concepts.

As in a factory, multiple assembly lines can operate independently of one another as they develop parts of the overall system, that are then assembled together to create finished products. Software APIs are analogous to the specifications and tolerances for parts that must fit together.

Like modern factories, automation plays a substantial part, and automation implies a degree of standardization and scale. Modern DevOps and Site Reliability Engineering approaches require infrastructure and tools that support a standardized approach with consistent performance at scale. The storage layer for such a system needs to support the processes and tools chosen without creating new headaches for the engineers tasked with building the automated software factory of the future.

Modern Data Access

Modern container-based software is developed in a distributed fashion, and uses data in a distributed fashion. Data may reside in one or more public clouds, within corporate data-centers on-site, or in co-location facilities. It is increasingly developed using heavily automated continuous integration and continuous deployment (CI/CD) techniques.

This data needs to be shared between the different, distributed teams so they can make use of it efficiently. It may be container images shared between different clusters, or large source datasets to be analyzed by multiple systems using different approaches.

Moving all this data around can be very time consuming with traditional extract/transform/load techniques.

With traditional, monolithic architectures, the complexities of distributed state management are mitigated by keeping the distribution to a minimum. While this reduces the complexity of system design, it creates silos of data that are not easily shared.

Coordinating state between multiple components, all of which are evolving rapidly in parallel, requires a solution that can scale horizontally as well as vertically, provide access to data in a variety of ways, and at a variety of locations. It must also be resilient to failures, lest containers lose access to the state they need. Multiple sites, locations and teams implies a need for multi-tenancy features, lest you end up implementing a collection of new data silos.

Access to data also needs to be efficient, hence the emphasis on logical partitions of data based on requirements rather than physical partitions. If the requirements change, so can the partitioning simply by changing access controls.

A good container storage solution needs to readily adapt to changes, because everything about this approach to software development and deployment is predicated on a constantly changing environment. Change is not an infrequent thing that is carefully meted out in discrete chunks, it is the constant background hum of the machine in operation.

Combination

The choices we make about data storage depend on the particular combination of requirements, rather than any one in isolation. We will often have multiple options as the data type we've identified will fall into multiple categories. It is therefore important to consider the changes in data that are likely to occur, not merely where we are today.

If we expect a significant increase in data volume, then we might make different choices than if we expect the volume to stay the same. Similarly, if we are confident the velocity of the data will remain static, we can ignore certain future options.

In making these assessments, we can never be 100% certain of what the future will hold. Instead, we need to remain conscious of the trade-offs we are making. If we decide on a certain option that cuts off future options, we should be mindful of the likelihood of being wrong, and the cost of changing our mind.

One of the great benefits of modern approaches is the reduction in friction and the cost of change. By keeping

future options open, we can reduce the cost of running experiments to test our theories about what works best. We can get rapid feedback on ideas, rather than spending months on detailed analysis only to be proven wrong by something we overlooked.

The Case For Filesystems

Given these requirements, we need a software coordination layer that can combine distributed storage devices into a logical pool of storage that is available for use by containers no matter where in a given cluster or pod they might be running. Data must be replicated to ensure it is accessible from any given node, and also to guard against node failures. We may also want to make use of additional data services like snapshots or deduplication.

Filesystems make an excellent choice for the vast middle ground of non-specialized storage systems. They are the default choice for data storage with containers partly for this reason. Filesystems provide a well-understood interface with a structure that is simple for humans to understand and manage, and both system administrators and developers understand them well. A filesystem is usually layered on top of some pool of raw storage to provide these higher-level functions.

Other options one might consider are:

- **Local storage, or traditional SAN and NAS technologies.** These can make good building blocks for the raw pools of storage needed--there has to be hardware here somewhere after all--but a distributed filesystem software layer is necessary to make this hardware truly usable in a container world.
- **Block storage services.** Block storage provides a good underpinning for filesystems and certain specialized storage systems. However, they do not readily permit shared access without some higher-level software to provide coordination mechanisms, and shared access to state is an important aspect of all applications.
- **Databases.** Databases provide another option for collecting raw storage together into a pool of shareable storage with additional data services layered on top, but they can be complex to administer, particularly at scale. Setting up a multi-master PostgreSQL cluster is not for the faint of heart.

  Databases are a good choice for highly structured data where the advantages of the specialized data access methods are worth the hassle of configuring and operating them. They are less compelling for more general purpose data storage.
- **Object stores.** Object stores provide another option, but are optimized for large quantities of immutable data with a high degree of read access, which is an even more specialized form of data storage than a general purpose database.

Filesystems provide a good compromise. In fact, many databases and object stores are implemented on top of a filesystem structure of some sort--or can be. Filesystems are flexible enough to be adapted to serve other, more specialized needs, by layering more specialized applications on top, and can be used simultaneously for other storage needs. Premature optimization can cause major hassles later when the system has to be reworked to move in a different direction than originally anticipated. The minor overhead of the additional layer is more than offset by the gains in flexibility and choice the layer provides, and the simplicity of having a single underlying technology choice for the majority of storage requirements.

Change and Transition

Container technologies, for all the enthusiasm surrounding them, are still a very new technology.

Using containers is a worthy goal, and they are definitely useful for a variety of situations and use-cases, but it is unreasonable to expect that an organization will replace all of its existing investments in infrastructure and applications with container-based solutions overnight. Existing systems need to be maintained alongside container systems, and a transition to 100% container-only is unlikely.

However, during the transition period, it would be useful to make existing data available to container systems. An investment in storage that supports both container development and use for traditional data allows a managed transition as organizational processes allow.

Such a system should provide a similar experience in both cloud and on-site deployments. Cloud environments are

ideally suited to experimentation, where multiple ideas and techniques can be tried, and resetting back to a zero state is easy, which parallels the benefits of containers. When an experiment succeeds, we want to be able to take the lessons from the experiment and apply them to other, more permanent environments, which is difficult if the cloud environment is too different.

Rather than having to go 'all in' on containers, a managed transition will allow teams to learn as they go, adapting to the new ways of working as they determine each new use-case that can benefit from containers.

Remember that the world of containers continues to change rapidly. Any decision made today is likely to require changes multiple times in the coming year, so flexibility is an important feature of your choice of container state management solution.

A filesystem approach provides the right blend of current functionality, backwards compatibility, and future flexibility.

Elastifile Cloud File System

The Elastifile Cloud File System (ECFS) provides the foundation for Elastifile's cross-cloud data fabric solution. The Elastifile solution, incorporating both ECFS and Elastifile's CloudConnect capability, aligns well with the requirements for management of stateful container data, particularly for organisations using a mix of cloud and on-site infrastructure.

Software Only Approach

The software-only nature of Elastifile's solution makes it easy to deploy in a variety of locations with relative ease. It can make use of cost-effective commodity servers connected to SSD storage or cloud-hosted flash-backed instances. Its scale-out architecture makes it easy to expand, or contract, as required and provides robust resilience against component or node failures.

Multi-Site and Hybrid Cloud

Elastifile's solution creates a single namespace to make data available across a deployment, even if it spans multiple physical sites and/or clouds. Organisations can use Elastifile to span production, test, and DR environments and have access to the same data at each. Each environment could be composed of different physical infrastructure, should that be desirable; there's no need for a logical test environment to exactly match that of production. When deployed in a hybrid or multi-cloud configuration, Elastifile leverages both ECFS and CloudConnect to provide an ideal solution for migrating and managing data between on-premises locations and the cloud, or between clouds.

Logical Containers

Elastifile provides a logical partitioning scheme with robust policy-based access controls: Data Containers. Each Data Container sits within the global namespace and provides isolation from other Data Containers based on the logical requirements of applications. Highly sensitive data can be kept separate from general purpose data without having to implement a completely different storage system. Policies can also be used to limit the physical data location should that be required, for example to keep sensitive data on-site, or limit its location to cloud deployments in specific countries.

Flash-Native Architecture

The Elastifile solution is designed to use flash. While spinning disk can provide a cost-effective way to store infrequently accessed bulk data, the continuously changing, online nature of containers is better suited to flash. Flash provides consistently lower latency, and much higher bandwidth, than spinning disk. At scale, low variance in operational variables becomes crucial, as the constantly changing nature of a container environment makes troubleshooting more complex. Making storage performance more predictable reduces overall system complexity, and promotes a more consistent user experience.

Elastifile provides a high-performance storage layer that will grow with you as you develop your container-based systems, providing choice and flexibility in this rapidly evolving field. No matter where you decide to use containers, on-site, in the cloud, or a mixture of both, the Elastifile approach provides a robust and feature-rich choice for managing state at scale.