Understanding digital twin environments

An overview of architecture and solution implementation

"IDC defines digital twins as virtual models of a product or asset connected to the physical prototype or instance via IoT. Digital twins visualize data flows and provide collaboration across engineering, operations, supply chains, and servicing."[1]

"Digital twin technology is an emerging concept that has become the center of attention for industry and, in more recent years, academia. The advancements in industry 4.0 concepts have facilitated its growth, particularly in the manufacturing industry."[2]

Overview of digital twin

Customers often use digital twins to experiment with software and simulated hardware components before committing to building a final version of the hardware or releasing a new version of software. Recently, digital twins have helped customers simulate how software changes can impact hardware or existing software components that have previously been released to the market. This impact is a key consideration when systems have a support lifetime measured in years or decades.

Red Hat has worked with multiple customers requiring digital twin solutions. This blueprint is a high-level outline of some of the architectural approaches we have used—and our lessons learned. It is not intended to be a definitive, one-size-fits-all approach, and it does not cover all aspects and use cases of a digital twin architecture. This solution has been built and validated in the manufacturing and automotive domain, but it is flexible enough to extend to many other domains, such as supervisory control and data acquisition (SCADA), aviation, Internet of Things (IoT), and financial services (FSI).

The challenge with digital twin environments

Digital twin environments can be expensive and complex to provision and deploy. Each environment is heterogeneous, making it difficult to replicate and scale. Changes and customization to these environments can be expensive and time consuming. Given the cost of the infrastructure, customers want to maximize the utilization of the underlying hardware and derive as much value out of the investment in building these environments. Therefore, a flexible solution is needed to create digital twin environments on demand to support a multitude of digital twin workloads.

Solution

This solution is based on requirements from customers who needed a fast, automatable, flexible, and reproducible mechanism for deploying digital twin environments to experiment with, simulate, or validate components (or collections of components) of a complex software system. For example, in the automotive space this solution was used to address two flavors of digital twin topologies. The first is a virtual approach where software components run in a fully self-contained environment, commonly referred to as Software-in-the-Loop (SIL).

The second is a hybrid approach, known as Hardware-in-the-Loop (HIL), where software components are run in a self-contained environment but also connect to the physical world where external hardware devices are incorporated into the solution space.

The solution is not just focused on the automation of a software or hardware setup—it also deals with the life cycle and control of the environments and their various third-party components.

facebook.com/redhatinc
@redhat
linkedin.com/company/red-hat
redhat.com

[1] Burian, Jan. "Digital Twins and Digital Threads: The Innovative Way to Track Product Life Cycles." IDC Blog. July 31, 2020.
[2] Fuller, Aidan, et al. "Digital Twin: Enabling Technologies, Challenges and Open Research." IEEE Access. Digital Object Identifier. May 28, 2020.

Digital twin environment

A digital twin environment (DTE) is a logical environment in which software and sometimes hardware components interact to simulate an entire system or subsystem. It can be used to test software component interactions or simulate scenarios to drive software components and record results. Components of DTEs vary depending on the customers use case, but they often include:

- Component simulators.
- Processing and rendering components.
- Signal and event streaming and handling.
- Data sources.
- Protocol gateways, such as Modbus, Distributed Network Protocol (DNP3), and Controller Area Network (Can Bus).
- Interaction and simulation scenarios.

A DTE may consist of one or more of these, or other, components. Many of these components may be sourced from multiple providers. A DTE definition is a versioned collection of these components and associated data.

The output from a DTE can vary according to the customer's use case but can include event or signal streams, logs, or videos. Often this output is processed by other external systems to interpret the results.

Common requirements for businesses when considering DTEs

Businesses examine the ability to:

- Provision, scale, and deprovision DTEs easily on demand and in an automated manner.
- Run multiple DTEs of similar or different types and versions simultaneously.
- Version control DTE definitions, software, and configuration values.
- Develop a standardized and consistent approach to building software components for DTEs.
- Use DTEs for a fixed period of time, after which they are automatically removed.
- Use application programming interfaces (APIs) to enable integration with existing customer workflows and tools.
- Build a standards-based solution that can be used on-premise and on public clouds.
- Develop a multitenant approach with multiple simultaneous users of the platform executing different DTE types.
- Use usage and user experience as a key requirement and work in the domain familiar to the user.

High-level architecture

The following solution consists of three major domains: control plane, data plane, and environment plane. The physical hardware domain shown in the diagram below is only necessary if components within the environment plane need to interact with physical hardware.

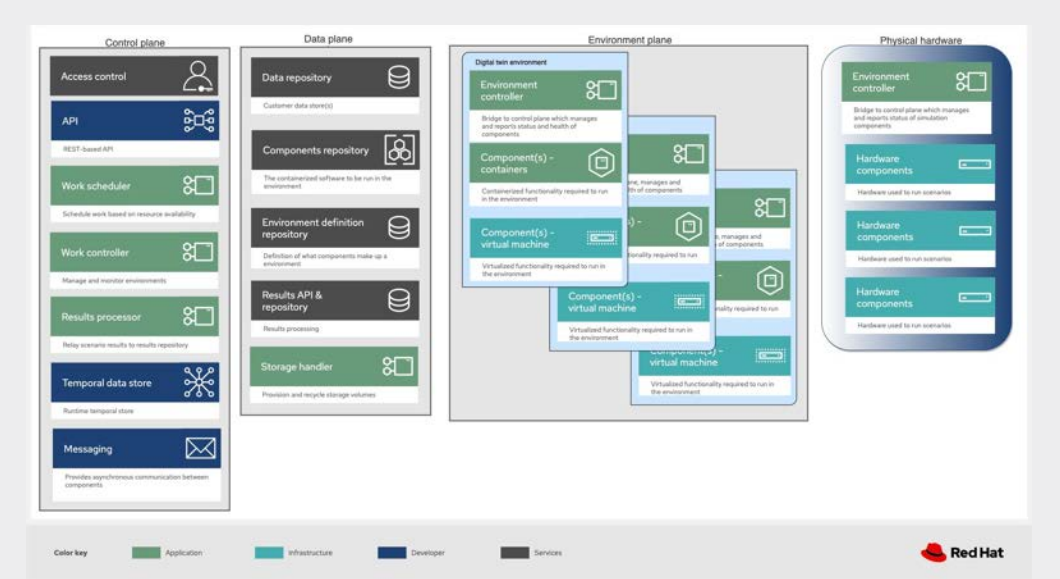

Figure 1. High-level digital twin environment architecture, detailing control plane, data plane, environment plane, and physical hardware, if needed

Control plane

The control plane (CP) is the primary entry point to the solution, responsible for:

- API handling for integration with user interface (UI) or other user workflow components (out of scope).
- Workload prioritization and scheduling.
- Life cycle and state management of individual DTEs.
- Event-driven interaction with individual DTEs, e.g., responsible for processing the events sent back from the various environment controllers and issuing commands to them as necessary (process data, run simulation, etc.).

Major components

- API gateway: receives, validates, and routes requests from external consumers.
- Access control: controls what user or system can access the platform and the visibility they have, e.g., DTE types and payloads, as well as the actions that they can perform.
- Work scheduler: contains business and scheduling rules to determine where and when a DTE will be created. Also manages and enforces limits on the resources used on the platform, such as the number of DTEs and external hardware usage.

- Work controller: handles the life cycle of a DTE. Receives and processes incoming status events from each environment. Contains logic for error handling and recovery, as well as DTE expiration handling.
- Temporal data store: replicated data store used for reliability storing temporal data, i.e., DTE events and DTE state. Used by the work scheduler and work controller.
- Results processor: handles results or output sent from a DTE.
- Messaging: used for asynchronous event-driven interaction between planes and components. Also used for work distribution across multiple DTEs if required.

Data plane

The data plane (DP) is the data management area of the solution. It is responsible for:

- Providing an API for storing and retrieval of digital twin related data, including:
  - Environment definitions.
  - Component configuration.
  - Simulation definitions.
  - Related data and event input streams.
  - Artificial Intelligence (AI) and data models.
- Providing an API for provisioning and pre-populating storage volumes to be used in an individual DTE.
- Providing version management of input and output artifacts.
- Providing interfaces/logic for result or output handling, as well as storage recycling.
- Interacting with existing customer data sources.
- Providing metadata management.

Major components

- Components repository: container image repository including containerized DTE components.
- Environment definition repository: a repository that stores the DTE definitions and configuration. It provides an API for create, read, update, and delete (CRUD) access.
- Storage handler: creates and populates volumes on the underlying storage platform for usage when the DTE is operational. Also performs cleanup and recycling of storage volumes when a DTE is terminated.
- Data repository: prepares data to be sent to the DTE (depending on type of processing). It is used in conjunction with the Customer Data Store.
- Results API: forwards results to downstream results processing systems. It is very specific to the customer use case.
- Customer Data Store: stores customer specific metadata and configuration used during the execution of the DTE, such as AI models or scenarios.

Physical hardware

The components that comprise the physical hardware environments are specific to customers' needs. Some of these environments are managed, meaning that components are controlled by an environment controller

that initializes and controls the environment components. Other environments are unmanaged, meaning that components are manually configured and controlled. For provisioning and automation purposes, the managed approach helps avoid time-consuming manual interaction.

Solution implementation

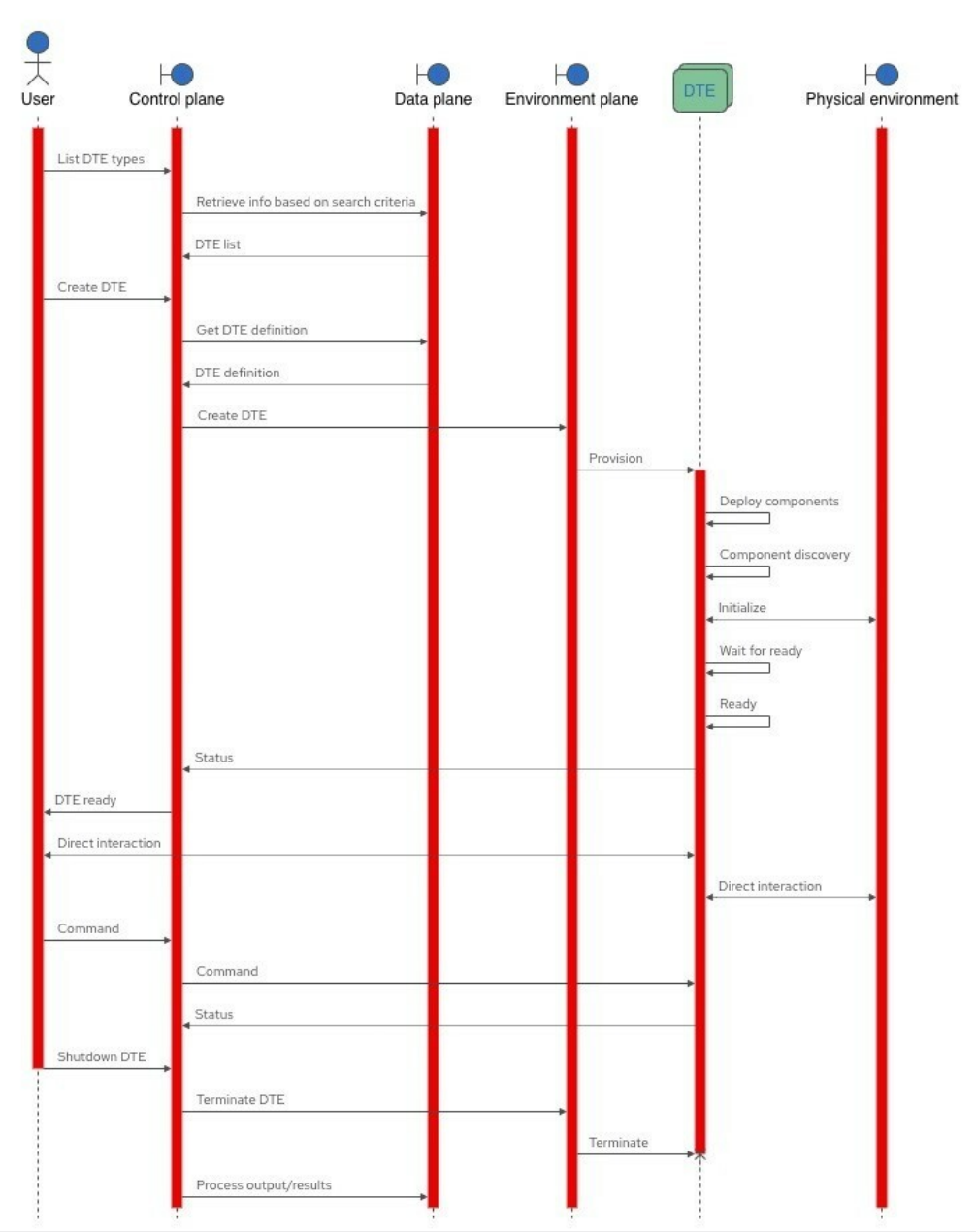

Figure 2. Digital twin environment solution implementation

Compute

The core of the solution revolves around containers and Kubernetes. The DTE components all run in containers, and Red Hat® OpenShift® is used to create and deploy DTEs based on customer needs. A DTE is implemented as a Red Hat OpenShift project. There are a number of benefits to this approach:

- Isolation: Red Hat OpenShift's multitenancy features are primarily based on projects and provide desirable features such as network isolation, which is key for component discovery and communication. Projects also support multicast network traffic isolated on a project-by-project basis that can be used for multicast discovery features of DTE components. Service discovery can also be performed via project-level DNS resolution.
- Unit of deployment: DTE components are deployed to a project and visible only within that project. As a result, multiple versions of components and different types of DTEs can be deployed.
- Project placement: Placement on nodes of certain DTE components may have explicit hardware requirement needs, such as graphic processing units (GPUs). Node selectors can be placed on projects, which results in pods being scheduled where their hardware requirements are fulfilled.
- Removing environments: Removing a project also results in all associated resources, e.g., deployments, secrets, and configmaps, being removed as well. This removal simplifies the DTE cleanup process, with the exception of storage, which has to be handled separately.

Red Hat OpenShift is also available on non x86 platforms like ARM, which is useful for components that are running on non x86-based architecture, e.g., electronic control units (ECUs) or remote telemetry unit (RTU) controllers. Microsoft Windows-based components can be run either using Red Hat OpenShift for Windows Containers or OpenShift Virtualization features. For workloads that may require advanced kernel isolation features, Red Hat OpenShift sandboxed containers are an option.

Many of the DTE components may require cluster-based hardware, such as graphics processing units (GPUs) or InfiniBand cards, but it is possible to use these components within Red Hat OpenShift as long as there is a vendor-supplied device manager[3] plugin available.

Network and communications

Network connectivity is a key consideration for digital twin solutions. Red Hat OpenShift supports the Multus Container Network Interface (CNI) plugin, which enables pods to have two or more network interfaces. This consideration is important as one network will be used to communicate with the DTE components and control/data planes, and the additional interfaces can be used to communicate to components hosted outside of the Red Hat OpenShift software-defined network (SDN) on separate networks, like external hardware devices. Red Hat OpenShift also supports IPV6 addressing and communications.

Communication between Red Hat OpenShift-based components and external hardware devices using non-IP based protocols (CanBus, DNP3, ModBus, etc.) requires protocol gateways to bridge between software components and physical hardware.

Command and events are transferred between control and data planes and DTE environment controllers over a flexible messaging protocol: AMQP 1.0. We use AMQP due to its asynchronous nature of communication, its flexible addressing model, and its standardization. AMQP 1.0 is an ISO standard ISO/IEC 19464:2014, which allows interaction with other compatible third-party components. Red Hat AMQ provides the AMQP compliant broker and client libraries.

[3] Red Hat documentation. "Using device plug-ins to access external resources with pods," accessed Dec. 17, 2021.

APIs and open standards

Application programming interfaces (APIs), open standards, and protocols are needed to ensure platform flexibility and adoption. The platform has to be flexible enough to run DTEs composed of multiple components. Many of these components use proprietary interfaces and protocols, which are encapsulated at the DTE boundary and container layer. As a result, a lot of complexity of the underlying system can be abstracted and only an opinionated selection of information is exposed to the user.

Data management and storage

A DTE solution uses a tremendous amount of data. Managing, storing, and controlling access to data is a substantial undertaking.

The primary sources of data used in our solution are:

- DTE ingress and egress data.
- DTE definitions and metadata storage.
- Component container images.
- Temporal data.

DTE ingress and egress data storage

Storage is critical given the large amount of data used and produced by workloads running in a DTE. Because many DTEs will execute in parallel with different data needs, different types of storage are required, including block, file, and object storage. The ability to dynamically create and mount storage for use within a DTE is a key concern in handling different types of DTEs. We use Red Hat OpenShift Data Foundation, which provides software-defined storage based on Ceph® technology.

Storage and distribution of streaming data, like events or signals, may also be required, and Red Hat AMQ Streams, which provides Apache Kafka clusters on Red Hat OpenShift, can be used for this purpose. This approach allows dynamic provisioning of Kafka topics or even entire Kafka clusters, depending on the customer's use case.

DTE definitions and metadata storage

A DTE definition is a collection of multiple components, both containerized and physical, as well as the configuration and data needed by these components. To control these moving parts, we recommend that all these artifacts be managed and accessed by version control. This approach builds on concepts such as software version control and infrastructure as code to ensure that DTEs are consistently reproducible and component and data changes are tracked.

There are many methods of storing this kind of information depending on the customer's use case (e.g., source code management (SCM) tools like Git, Open Container Initiative (OCI)-compliant repositories, or custom repositories).

Temporal data

The control plane components receive many status events from DTEs. These events are used to update the internal view of the DTE's state and manage the life cycle of the DTE. This state view needs to be resilient in the case of a control plane component error, but given the temporal and ever-changing nature of this state, storing this data in a database is not a suitable approach. Instead, the materialized state of DTEs and clustered components is stored in a reliable distributed data grid, which provides the resiliency needed if a control plane fails. Red Hat Data Grid running on Red Hat OpenShift provides this functionality.

Component container images

Red Hat OpenShift comes with its own internal container registry. However, it is specifically for run time usage. The solution needs to be flexible enough for external organizations and users to upload container images to run as part of a DTE. These images need to be access controlled and scanned for security purposes. We use Red Hat Quay, a container image registry, to meet this need.

Access control

Many organizations using a digital twin approach have concerns about intellectual property leakage and security. There may be competing users of the platform, and even though access control and workload isolation is present on the Red Hat OpenShift platform, it needs to be extended across the entire solution. Determine who can see and use specific data and what DTE definitions can be used. Most of the data resources we have used in customer environments are accessible via secured APIs. Modern API access control protocols, such as OpenID Connect (OIDC), can be used to control access to these resources. Legacy systems can also be adapted using a facade approach to unify access control. However, experience tells us that access control is often a nuanced and complex collection of policies, groups, and permissions.

Red Hat's single sign-on (SSO) technology centralizes access control to APIs via its fine-grained authorization policies and policy enforcement functionality.

Red Hat OpenShift's multitenancy and isolation features ensure that security is enforced between DTEs and users can only access DTEs for which they have explicit permission.

Scheduling workloads

For many types of DTEs, the Kubernetes Scheduler in Red Hat OpenShift is sufficient for scheduling workloads with its built-in support for scheduling containers based on cluster resources. However, when scheduling extends to non cluster-based resources (such as external physical hardware and workload priorities) then an external scheduler is required to balance the resource needs. When a more holistic scheduler is required, we use the flexible OptaPlanner scheduler functionality within Red Hat Decision Manager.

Component diagram

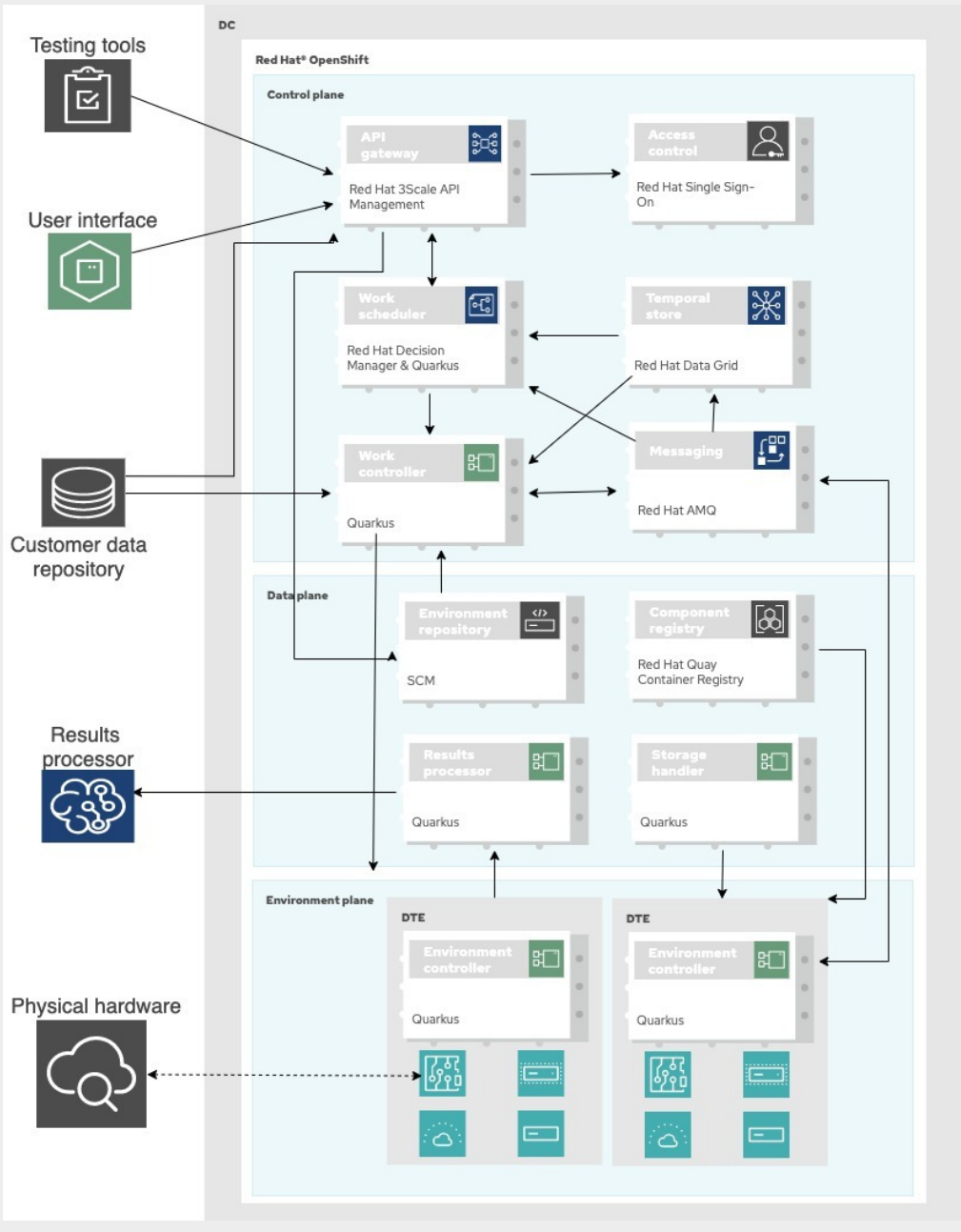

Figure 3. Digital twin environment component diagram, including testing tools, user interface, customer data repository, results processor, and physical hardware

Conclusion

As digital twin technology continues to advance, organizations can benefit by testing simulated hardware components to help protect their investment in building these environments.

To learn more about how Red Hat OpenShift can help organizations build and manage digital twin environments effectively, contact Red Hat Open Innovation Labs.

NORTH AMERICA
1 888 REDHAT1
www.redhat.com

EUROPE, MIDDLE EAST, AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@redhat
linkedin.com/company/red-hat

redhat.com
O-F30893