









Java EE application components

The product supports application components that conform to Java Platform, Enterprise Edition (Java EE) specifications.

**Web applications run in the web container**

The web container is the part of the application server in which web application components run. Web applications are comprised of one or more related servlets, JavaServer Pages technology (JSP files), and Hyper Text Markup Language (HTML) files that you can manage as a unit. Combined, they perform a business logic function.

The web container processes servlets, JSP files, and other types of server-side includes. Each application server runtime has one logical web container, which can be modified, but not created or removed. Each web container provides the following.

**Web container transport chains**

Requests are directed to the web container using the web container inbound transport chain. The chain consists of a TCP inbound channel that provides the connection to the network, an HTTP inbound channel that serves HTTP requests, and a web container channel over which requests for servlets and JSP files are sent to the web container for processing.

**Servlet processing**

When handling servlets, the web container creates a request object and a response object, then invokes the servlet service method. The web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

*Servlets* can perform such tasks as supporting dynamic web page content, providing database access, serving multiple clients at one time, and filtering data.

*JSP files* enable the separation of the HTML code from the business logic in web pages. IBM® extensions to the JSP specification make it easy for HTML authors to add the power of Java technology to web pages, without being experts in Java programming.

**HTML and other static content processing**

Requests for HTML and other static content that are directed to the web container are served by the web container inbound chain. However, in most cases, using an external web server and web server plug-in as a front end to the web container is more appropriate for a production environment.

**Session management**

Support is provided for the javax.servlet.http.HttpSession interface as described in the Servlet application programming interface (API) specification.

An *HTTP session* is a series of requests to a servlet, originating from the same user at the same browser. Sessions allow applications running in a web container to keep track of individual users. For example, many web applications allow users to dynamically collect data as they move through the site, based on a series of selections on pages they visit. Where the user goes next, or what the site displays next, might depend on what the user has chosen previously from the site. To maintain this data, the application stores it in a “session”.

**SIP applications and their container**

SIP applications are Java programs that use at least one Session Initiation Protocol (SIP) servlet. SIP is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging.

**Portlet applications and their container**

Portlet applications are special reusable Java servlets that appear as defined regions on portal pages. Portlets provide access to many different applications, services, and web content.

**EJB applications run in the EJB container**

The EJB container provides all of the runtime services needed to deploy and manage enterprise beans. It is a server process that handles requests for both session and entity beans.

*Enterprise beans* are Java components that typically implement the business logic of Java EE applications, as well as accessing data. The enterprise beans, packaged in EJB modules, installed in an application server do not communicate directly with the server. Instead, the EJB container is an interface between EJB components and the application server. Together, the container and the server provide the enterprise bean runtime environment.

The container provides many low-level services, including threading and transaction support. From an administrative perspective, the container handles data access for the contained beans. A single container can host more than one EJB Java archive (JAR) file.

Client applications and other types of clients

In a client-server environment, clients communicate with applications running on the server. *Client applications* or *application clients* generally refers to clients implemented according to a particular set of Java specifications, and which run in the client container of a Java EE-compliant application server. Other clients in the WebSphere Application Server environment include clients implemented as web applications (*web clients*), clients of web services programs (*web services clients*), and clients of the product systems administration (*administrative clients*).

**Client applications and their container**

The client container is installed separately from the application server, on the client machine. It enables the client to run applications in an EJB-compatible Java EE environment. The diagram shows a Java client running in the client container.

This product provides a convenient [launchClient tool](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/rcli_javacmd.html) for starting the application client, along with its client container runtime.

Depending on the source of technical information, client applications sometimes are called application clients. In this documentation, the two terms are synonymous.

**Web clients, known also as web browser clients**

The diagram shows a web browser client, which can be known simply as a web client, making a request to the web container of the application server. A web client or web browser client runs in a web browser, and typically is a web application.

**Web services clients**

Web services clients are yet another kind of client that might exist in your application serving environment. The diagram does not depict a web services client. The web services information includes information about this type of client.

**Administrative clients**

The diagram shows two kinds of administrative clients: a scripting client and the administrative console that is the graphical user interface (GUI) for administering this product. Both are accessing parts of the systems administration infrastructure. In the sense that they are basically the same for whatever kind of applications you are deploying on the server, administrative clients are part of the product architecture. However, because many of these clients are programs you create, they are discussed as part of the programming model for completeness.

See [Using the administrative clients](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/welc_adminclients.html).

Web services

**Web services**

The diagram shows the web services engine, part of the web services support in the application server runtime. Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a service-oriented architecture (SOA), which supports the connecting or sharing of resources and data in a flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

The product acts as both a web services provider and as a requestor. As a provider, it hosts web services that are published for use by clients. As a requester, it hosts applications that invoke web services from other locations. The diagram shows the web services engine in this capacity, contacting a web services provider or gateway.

Data access, messaging, and Java EE resources

**Data access resources**

Connection management for access to enterprise information systems (EIS) in the application server is based on the Java EE Connector Architecture (JCA) specification. The diagram shows JCA services helping an application to access a database in which the application retrieves and persists data.

The connection between the enterprise application and the EIS is done through the use of EIS-provided resource adapters, which are plugged into the application server. The architecture specifies the connection management, transaction management, and security contracts between the application server and EIS.

The Connection Manager (not shown) in the application server pools and manages connections. It is capable of managing connections obtained through both resource adapters defined by the JCA specification and data sources defined by the JDBC 2.0 Extensions specification.

*JDBC resources* (JDBC providers and data sources) are a type of *Java EE resource* used by applications to access data. Although data access is a broader subject than that of JDBC resources, this information often groups data access under the heading of Java EE resources for simplicity.

*JCA resource adapters* are another type of Java EE resource used by applications. The JCA defines the standard architecture for connecting the Java EE platform to heterogeneous EIS. Imagine an ERP, mainframe transaction processing, database systems, and legacy applications not written in the Java programming language.

The JCA resource adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the connectivity between Java EE application servers or clients and an EIS. To use a resource adapter, install the resource adapter code and create configurations that use that adapter. The product provides a predefined relational resource adapter for your use.

**Messaging resources and messaging engines**

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Applications can use message-driven beans to automatically retrieve messages from JMS destinations and JCA endpoints without explicitly polling for messages.

For inbound non-JMS requests, message-driven beans use a Java EE Connector Architecture (JCA) 1.5 resource adapter written for that purpose. For JMS messaging, message-driven beans can use a JCA-based messaging provider such as the default messaging provider that is part of the product.

The messaging engine supports the following types of message providers.

**Default messaging provider (service integration bus)**

The default messaging provider uses the service integration bus for transport. The default message provider provides point-to-point functions, as well as publish and subscribe functions. Within this provider, you define JMS connection factories and destinations that correspond to service integration bus destinations.

**IBM MQ provider**

You can use IBM MQ as the external JMS provider. The application server provides the JMS client classes and administration interface, while IBM MQ provides the queue-based messaging system.

**Generic JMS provider**

You can use another messaging provider as long as it implements the ASF component of the JMS 1.0.2 specification. JMS resources for this provider cannot be configured using the administrative console.

**For transitioning users:** Version 6 replaces the Version 5 concept of a JMS server with a messaging engine built into the application server, offering the various kinds of providers mentioned previously. The Version 5 messaging provider is offered for configuring resources for use with Version 5 embedded messaging. You also can use the Version 5 default messaging provider with a service integration bus.

EJB 2.1 introduces an ActivationSpec for connecting message-driven beans to destinations. For compatibility with Version 5, you still can configure JMS message-driven beans (EJB 2.0) against a listener port. For those message-driven beans, the message listener service provides a listener manager that controls and monitors one or more JMS listeners, each of which monitors a JMS destination on behalf of a deployed message-driven bean.

**Service integration bus**

The service integration bus provides a unified communication infrastructure for messaging and service-oriented applications. The service integration bus is a JMS provider that provides reliable message transport and uses intermediary logic to adapt message flow intelligently into the network. It supports the attachment of web services requestors and providers. Its capabilities are fully integrated into product architecture, including the security, system administration, monitoring, and problem determination subsystems.

The service integration bus is often referred to as just a bus. When used to host JMS applications, it is often referred to as a messaging bus. It consists of the following parts (not shown at this level of detail in the diagram).

**Bus members**

Application servers added to the bus.

**Messaging engine**

The component that manages bus resources. It provides a connection point for clients to produce or from where to consume messages.

**Destinations**

The place within the bus to which applications attach to exchange messages. Destinations can represent web services endpoints, messaging point-to-point queues, or messaging publish and subscribe topics. Destinations are created on a bus and hosted on a messaging engine.

**Message store**

Each messaging engine uses a set of tables in a supported data store (such as a JDBC database) to hold information such as messages, subscription information, and transaction states.

Through the service integration bus web services enablement, you can:

* Make an internal service that is already available at a service destination available as a web service.
* Make an external web service available at a service destination.
* Use the web services gateway to map an existing service, either an internal service or an external web service, to a new web service that appears to be provided by the gateway.

**Concurrency, Mail, URLs, and other Java EE resources**

The following kinds of Java EE resources are used by applications deployed on a J2EE-compliant application server.

* JDBC resources and other technology for data access (previously discussed)
* JCA resource adapters (previously discussed)
* JMS resources and other messaging support (previously discussed)
* Concurrency resources for submitting or scheduling tasks to run in parallel, creating threads that inherit Java EE context, and transferring Java EE context to invocation of interfaces such as asynchronous callbacks
* JavaMail support, for applications to send Internet mail

The **JavaMail APIs** provide a platform and protocol-independent framework for building Java-based mail client applications. The APIs require service providers, known as protocol providers, to interact with mail servers that run on the appropriate protocols.

A mail provider encapsulates a collection of protocol providers, including Simple Mail Transfer Protocol (SMTP) for sending mail; Post Office Protocol (POP) for receiving mail; and Internet Message Access Protocol (IMAP) as another option for receiving mail. To use another protocol, you must install the appropriate service provider for the protocol.

JavaMail requires not only service providers, but also the JavaBeans Activation Framework (JAF), as the underlying framework to handle complex data types that are not plain text, such as Multipurpose Internet Mail Extensions (MIME), URL pages, and file attachments.

* URLs, for describing logical locations

URL providers implement the functionality for a particular URL protocol, such as HTTP, enabling communication between the application and a URL resource that is served by a particular protocol. A default URL provider is included for use by any URL resource with protocols based on the supported Java Platform, Standard Edition (Java SE) specification, such as HTTP, FTP, or File. You also can plug in your own URL providers that implement additional protocols.

* Resource environment entries, for mapping logical names to physical names

The java:comp/env environment provides a single mechanism by which both the JNDI name space objects and local application environment objects can be looked up. The product provides numerous local environment entries by default.

The Java EE specification also provides a mechanism for defining customer environment entries by defining entries in the standard deployment descriptor of an application. The Java EE specification uses the following methods to separate the definition of the resource environment entry from the application.

* + Requiring the application server to provide a mechanism for defining separate administrative objects that encapsulate a resource environment entry. The administrative objects are accessible using JNDI in the application server local name space (java:comp/env).
  + Specifying the administrative object's JNDI lookup name and expected returned object type. This specification is performed in the aforementioned resource environment entry in the deployment descriptor.

The product supports the use of resource environment entries with the following administrative concepts.

* + A *resource environment entry* defines the binding target (JNDI name), factory class, and return object type (via the link to a referenceable) of the resource environment entry.
  + A *referenceable* defines the class name of the factory that returns object instances implementing a Java interface.
  + A *resource environment provider* groups together the referenceable, resource environment entries and any required custom properties.

Security

**Security programming model and infrastructure**

The product provides security infrastructure and mechanisms to protect sensitive Java EE resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability.

Security infrastructure and mechanisms protect Java Platform, Enterprise Edition (Java EE) resources and administrative resources, addressing your enterprise security requirements. In turn, the security infrastructure of this product works with the existing security infrastructure of your multiple-tier enterprise computing framework. Based on open architecture, the product provides many plug-in points to integrate with enterprise software components to provide end-to-end security.

The security infrastructure involves both a programming model and elements of the product architecture that are independent of the application type.

Additional services for use by applications

**Naming and directory**

Each application server provides a naming service that in turn provides a Java Naming and Directory Interface (JNDI) name space. The service is used to register resources hosted on the application server. The JNDI implementation is built on top of a Common Object Request Broker Architecture (CORBA) naming service (CosNaming).

JNDI provides the client-side access to naming and presents the programming model used by application developers. CosNaming provides the server-side implementation and is where its name space is actually stored. JNDI essentially provides a client-side wrapper of the name space stored in CosNaming, and interacts with the CosNaming server on behalf of the client.

Clients of the application server use the naming architecture to obtain references to objects related to those applications. The objects are bound into a mostly hierarchical structure called the name space. It consists of a set of name bindings, each one of which is a name relative to a specific context and the object bound with that name. The name space can be accessed and manipulated through a name server.

This product provides the following naming and directory features.

* Distributed name space, for additional scalability
* Transient and persistent partitions, for binding at various scopes
* Federated name space structure across multiple servers
* Configured bindings for defining bindings bound by the system at server startup
* Support for CORBA Interoperable Naming Service (INS) object URLs

Note that with the addition of virtual member manager to provide federated repository support for product security, the product now offers more extensive and sophisticated identity management capabilities than ever before, especially in combination with other WebSphere and Tivoli® products.

**Object Request Broker (ORB)**

The product uses an ORB to manage interaction between client applications and server applications, as well as among product components. An ORB uses IIOP to enable clients to make requests and receive requests from servers in a network distributed environment.

The ORB provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client, providing location transparency.

Although not shown in the diagram, one place in which the ORB comes into play is where the client container is contacting the EJB container on behalf of a Java client.

**Transactions**

Part of the application server is the transaction service. The product provides advanced transactional capabilities to help application developers avoid custom coding. It provides support for the many challenges related to integrating existing software assets with a Java EE environment. These measures include ActivitySessions.

Applications running on the server can use transactions to coordinate multiple updates to resources as one unit of work such that all or none of the updates are made permanent. Transactions are started and ended by applications or the container in which the applications are deployed.

The application server is a transaction manager that supports coordination of resource managers and participates in distributed global transactions with other compliant transaction managers.

The server can be configured to interact with databases, JMS queues, and JCA connectors through their local transaction support when distributed transaction support is not required.

How applications use transactions depends on the type of application, for example:

* A session bean either can manage its transactions itself, or delegate the management of transactions to the container.
* Entity beans use container-managed transactions.
* Web components, such as servlets, use bean-managed transactions.

The product handles transactions with the following components.

* A transaction manager supports the enlistment of recoverable XAResources and ensures each resource is driven to a consistent outcome, either at the end of a transaction, or after a failure and restart of the application server.
* A container manages the enlistment of XAResources on behalf of deployed applications when it performs updates to transactional resource managers such as databases. Optionally, the container can control the demarcation of transactions for EJB applications that have enterprise beans configured for container-managed transactions.
* An API handles bean-managed enterprise beans and servlets, allowing such application components to control the demarcation of their own transactions.

WebSphere extensions

WebSphere programming model extensions are the programming model benefits you gain by purchasing this product. They represent leading edge technology to enhance application capability and performance, and make programming and deployment faster and more productive.

In addition, your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis. The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers who build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application. For more information about this feature, see [Application extension registry](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/cweb_extensions.html).

The various WebSphere programming model extensions, and the corresponding application services that support them in the application server runtime, can be considered in three groups: Business Object Model extensions, Business Process Model extensions, and extensions for producing Next Generation Applications.

Extensions pertaining to the Business Object Model

Business object model extensions operate with business objects, such as enterprise bean (EJB) applications.

**Application profiling**

Application profiling is a WebSphere extension for defining strategies to dynamically control concurrency, prefetch, and read-ahead.

Application profiling and access intent provide a flexible method to fine-tune application performance for enterprise beans without impacting source code. Different enterprise beans, and even different methods in one enterprise bean, can have their own intent to access resources. Profiling the components based on their access intent increases performance in the application server runtime.

**Dynamic query**

Dynamic query is a WebSphere programming extension for unprecedented application flexibility. It lets you dynamically build and submit queries that select, sort, join, and perform calculations on application data at runtime. Dynamic Query service provides the ability to pass in and process EJB query language queries at runtime, eliminating the need to hard-code required queries into deployment descriptors during application development.

Dynamic query improves enterprise beans by enabling the client to run custom queries on EJB components during runtime. Until now, EJB lookups and field mappings were implemented at development time and required further development or reassembly in order to be changed.

**Dynamic cache**

The dynamic cache service improves performance by caching the output of servlets, commands, and JSP files. This service within the application server intercepts calls to cacheable objects and either stores the output of the object or serves the content of the object from the dynamic cache.

Because Java EE applications have high read-write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create opportunity for significant gains in server response time, throughput, and scalability.

Features include cache replication among clusters, cache disk offload, Edge side include caching, and external caching - the ability to control caches outside of the application server, such as that of your Web server.

Extensions pertaining to the Business Process Model

Business process model extensions provide process, workflow functionality, and services for the application server. Use them in conjunction with business integration capabilities.

**ActivitySessions**

ActivitySessions are a WebSphere extension for reducing the complexity of dealing with commitment rules and limitations associated with one-phase commit resources.

ActivitySessions provide the ability to extend the scope of multiple local transactions, and to group them. This enables them to be committed based on deployment criteria or through explicit program logic.

**Web services**

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

Extensions for creating next generation applications

*Next generation extentions* can be used in applications that need the specific extensions. These enable next generation development by leveraging the latest innovations that build on today's Java EE standards. This provides greater control over application development, execution, and performance than was ever possible before.

**Asynchronous beans**

Asynchronous Beans is deprecated and is replaced by Concurrency Utilities for Java EE.

Asynchronous beans offer performance enhancements for resource-intensive tasks by enabling single tasks to run as multiple tasks. Asynchronous scheduling facilities can also be used to process parallel processing requests in “batch mode” at a designated time. The product provides full support for asynchronous execution and invocation of threads and components within the application server. The application server provides execution and security context for the components, making them an integral part of the application.

**Startup beans**

Startup beans allow the automatic execution of business logic when the application server starts or stops. For example, they might be used to pre-fill application-specific caches, initialize application-level connection pools, or perform other application-specific initialization and termination procedures.

**Object pools**

Object pools provide an effective means of improving application performance at runtime, by allowing multiple instances of objects to be reused. This reuse reduces the overhead associated with instantiating, initializing, and garbage-collecting the objects. Creating an object pool allows an application to obtain an instance of a Java object and return the instance to the pool when it has finished using it.

**Internationalization**

The internationalization service is a WebSphere extension for improving developer productivity. It allows you to automatically recognize the time zone and location information of the calling client, so that your application can act appropriately. The technology enables you to deliver each user, around the world, the correct date and time information, the appropriate currencies and languages, and the correct date and decimal formats.

**Scheduler**

The scheduler service is a WebSphere programming extension responsible for starting actions at specific times or intervals. It helps minimize IT costs and increase application speed and responsiveness by maximizing utilization of existing computing resources. The scheduler service provides the ability to process workloads using parallel processing, set specific transactions as high priority, and schedule less time-sensitive tasks to process during low traffic off-hours.

**Work areas**

Work areas are a WebSphere extension for improving developer productivity. Work areas provide a capability much like that of “global variables”. They provide a solution for passing and propagating contextual information between application components.

Work areas enable efficient sharing of information across a distributed application. For example, you might want to add profile information as each customer enters your application. By placing this information in a work area, it will be available throughout your application, eliminating the need to hand-code a solution or to read and write information to a database.

* [**WebSphere Application Server roles and goals**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/rwas_swg_roles.html)  
  There are several different computing roles that members of your organization might undertake when working with WebSphere Application Server.
* [**Fast paths for WebSphere Application Server**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/tovr_devphases.html)  
  Use the fast paths to deploy applications quickly and easily. The links go to topics that pinpoint the relevant information for reaching your goals quickly. The fast paths are intended to help you gain a little experience. The fast paths do not showcase the advanced product features that some users need or want to use in their production environments.
* [**Product overview**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_arch.html)  
  Learn about WebSphere Application Server.
* [**Java SE 8 in WebSphere Application Server traditional V9**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_javase8.html)  
  WebSphere Application Server traditional is adding support for Java Platform, Standard Edition 8, (Java SE 8) as its default platform support. The IBM SDK, Java Technology Edition, Version 8 is the implementation of Java SE 8 that WebSphere Application Server traditional Version 9.0 uses and it is fully compatible with Oracle Java SE version 8 libraries.
* [**Java EE 7 in WebSphere Application Server traditional**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_javaee7.html)  
  WebSphere Application Server traditional supports the full Java Platform, Enterprise Edition (Java EE) 7.
* [**Open source software APIs**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/opensourcesoftwareapis.html)  
  The WebSphere Application Server product includes a variety of open source software packages.
* [**Programming model APIs and specifications**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/rovr_specs.html)  
  The product supports various industry standards. Browse the lists of technologies to see what application programming interfaces (APIs) and specifications the WebSphere Application Server products support.
* [**Three-tier architectures**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_3-tier.html)  
  WebSphere Application Server provides the application logic layer in a three-tier architecture, enabling client components to interact with data resources and legacy applications.
* [**Development and assembly tools**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/catk_assemblytools.html)  
  You can use an Integrated Development Environment to develop, assemble, and deploy Java Platform, Enterprise Edition (Java EE) modules for WebSphere Application Server.
* [**Accessing the samples**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_samples.html)  
  The product offers samples that demonstrate common enterprise application tasks. Many samples also provide instructions for deployment and coding examples.
* [**Multimedia**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_media.html)  
  Watch videos, webcasts, and other media about WebSphere Application Server.
* [**Resources for learning**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/welc_rfl.html)  
  This topic familiarizes you with the many websites containing technical information for understanding and using your WebSphere Application Server product. A wealth of online information is available to complement the product documentation.
* [**Tutorials**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_tutorials.html)  
  Tutorials contain educational materials to help you learn the technologies and concepts behind products. Use tutorials and their accompanying samples to learn how to accomplish your goals with the product.
* [**WebSphere platform and related software**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/covr_family.html)  
  Learn about other useful WebSphere and IBM products.
* [**Viewing documentation offline**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/tovr_view_doc_offline.html)  
  You can download product documentation that is shown online to your computer and view it offline.
* [**WebSphere Application Server Considerations for GDPR readiness**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/was_gdpr.html)  
  Consider information about WebSphere Application Server features that you can configure and aspects of product use so that you can prepare your organization for General Data Protection Regulation (GDPR) readiness.
* [**Privacy Policy Considerations**](https://www.ibm.com/docs/en/SSEQTP_9.0.5/com.ibm.websphere.base.doc/ae/privacy_was.html)