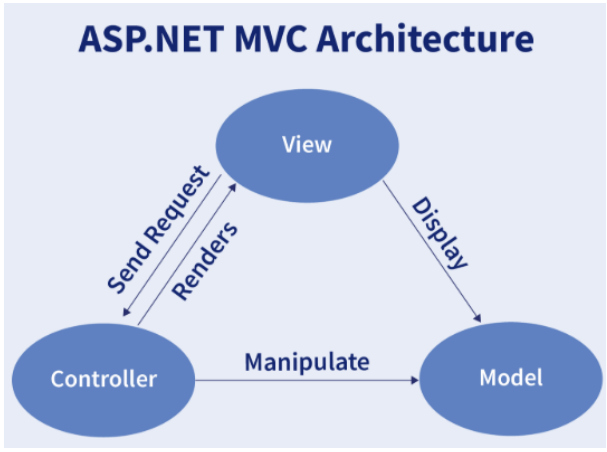

110+ Asp.Net Core MVC

Interview Questions and Answers

Sai Reddy

6/18/25

Interview Tips

110+ ASP.Net Core MVC Interview Q&A

0–2 years ASP.NET Core MVC interview questions:

1. What is ASP.NET Core MVC?

ASP.NET Core MVC is a framework for building web apps using the Model-View-Controller pattern.
It supports clean separation of concerns, testability, and routing.
It combines Razor view engine and web APIs into one platform.
MVC stands for Model (data), View (UI), and Controller (logic).
It runs cross-platform and is lightweight.

2. Difference between ASP.NET MVC and ASP.NET Core MVC?

ASP.NET Core MVC is cross-platform and modular.
It unifies Web API and MVC, unlike ASP.NET MVC which separates them.
ASP.NET Core uses the Startup.cs file and DI by default.
It's faster, lightweight, and open source.
ASP.NET Core runs on .NET Core or .NET 5+.

3. Role of Startup.cs file?

The Startup.cs file configures services and request pipeline.
It contains ConfigureServices() to register dependencies.
It has Configure() to define HTTP request processing.
Used during app startup by the runtime.

```
public void ConfigureServices(IServiceCollection services) {
   services.AddControllersWithViews();
}
```

4. What is the use of appsettings.json?

It stores app configuration (e.g., connection strings, logging).
Supports hierarchical JSON format.
Can load values using IConfiguration.
Supports environment-specific config like appsettings.Development.json.

```
{
  "ConnectionStrings": {
    "Default": "Server=.;Database=MyDB;Trusted_Connection=True;"
  }
}
```

5. What is middleware in ASP.NET Core?

Middleware are components in the request pipeline.
They handle or modify requests/responses.
Each middleware can pass to the next or short-circuit.
Configured in Configure() method.

```
app.UseRouting();
app.UseAuthorization();
```

6. What is the MVC pattern?

MVC stands for Model, View, and Controller.
Model holds data, View is the UI, Controller handles logic.
It separates concerns in application development.
Helps in testability and maintainability.
Controller -> Model -> View

7. How is routing handled in ASP.NET Core MVC?

Routing maps URLs to controller actions.
It's configured in Program.cs or Startup.cs.
Attribute routing and conventional routing are supported.

```
app.MapControllerRoute(
  name: "default",
  pattern: "{controller=Home}/{action=Index}/{id?}");
```

8. What is dependency injection in ASP.NET Core?

DI provides class dependencies from a central container.
Built-in in ASP.NET Core using services.Add....
Improves testability and code quality.

```
services.AddScoped<IRepo, Repo>();
```

9. How do you return JSON from a controller action?

Use JsonResult or return object directly with [ApiController].

```
public JsonResult GetData() {
   return Json(new { id = 1, name = "John" });
}
```

10. What are tag helpers in ASP.NET Core?

Tag helpers enable server-side rendering using HTML-like syntax.
They improve readability and maintainability of Razor views.

```
<form asp-controller="Home" asp-action="Login"></form>
```

11. What is the purpose of the ConfigureServices method?

It registers services into the DI container.
Called by the runtime before the app runs.

```
public void ConfigureServices(IServiceCollection services) {
   services.AddControllersWithViews();
}
```

12. How do you perform form validation in ASP.NET Core MVC?

Use data annotations on model properties.
Add validation scripts in view.

```
[Required]
public string Name { get; set; }
```

13. What are Razor views?

Razor views (.cshtml) are HTML + C# syntax for dynamic UI.
Use @ to embed C# code in HTML.

<h1>Hello @Model.Name</h1>

14. What is ViewBag, ViewData, and TempData?

They pass data from controller to view.

- ViewBag: dynamic object
- ViewData: dictionary
- TempData: persists across redirects

ViewBag.Message = "Hi";
TempData["Status"] = "Saved";

15. How do you create a strongly typed view?

Use @model in Razor view.
Pass model from controller.
return View(new Product());
@model Product
@Model.Name

16. IActionResult vs ActionResult

- IActionResult: multiple return types
- ActionResult<T>: combines result + model type

public IActionResult Index() => View();
public ActionResult<Product> Get() => product;

17. AddScoped vs AddTransient vs AddSingleton

- Scoped: per request
- Transient: new instance every time
- Singleton: one instance for app

services.AddScoped<IMyService, MyService>();

18. How do you handle errors in ASP.NET Core MVC?

Use app.UseExceptionHandler("/Home/Error").
Can create custom error pages or logging.
app.UseExceptionHandler("/Error");

19. What are partial views and when to use them?

Partial views are reusable UI components.
Used for header, footer, form sections.
@Html.Partial("_LoginPartial")

20. What is model binding in ASP.NET Core?

It maps form/query/route data to method parameters.
Used automatically in controller actions.
public IActionResult Save(Product p) { }

21. What is model validation?

Checks if model state is valid using attributes.
Check ModelState.IsValid in controller.
if (!ModelState.IsValid) return View(model);

22. What is the _Layout.cshtml file?
Defines shared layout (like master page).
Use @RenderBody() for page content.
<!DOCTYPE html><body>@RenderBody()</body>

23. What is ViewComponent?
Reusable rendering logic, similar to partial view but with C# class.
public class BannerViewComponent : ViewComponent {
  public IViewComponentResult Invoke() => View();
}

24. How to use session in ASP.NET Core MVC?
Add session middleware and services.
services.AddSession();
app.UseSession();
HttpContext.Session.SetString("Key", "Val");

25. How do you use static files in ASP.NET Core?
Add app.UseStaticFiles(); in Configure().
Place files in wwwroot folder.

26. What is the [Bind] attribute used for?
Limits model binding to specific properties.
public IActionResult Save([Bind("Id,Name")] Product p) { }

27. How do you post a form in ASP.NET Core MVC?
Create form in view and accept model in POST action.
<form asp-action="Save" method="post">...</form>
[HttpPost]
public IActionResult Save(Product p) { }

28. What is AntiForgeryToken and why is it needed?
Prevents CSRF attacks.
Add @Html.AntiForgeryToken() in form.
Check with [ValidateAntiForgeryToken].
<form method="post">@Html.AntiForgeryToken()</form>

29. What are action filters?
Used to run code before/after action executes.
Inherit ActionFilterAttribute.
public class LogFilter : ActionFilterAttribute { ... }

30. How do you implement file upload in ASP.NET Core MVC?
Use IFormFile in POST action.
<form enctype="multipart/form-data" method="post">
  <input type="file" name="file" />

```
</form>
[HttpPost]
public async Task<IActionResult> Upload(IFormFile file) {
  var path = Path.Combine("uploads", file.FileName);
  using var stream = new FileStream(path, FileMode.Create);
  await file.CopyToAsync(stream);
  return Ok();
}
```

3–6 Years Experience (Intermediate)

1. How does ASP.NET Core support cross-platform development?

ASP.NET Core runs on .NET Core which is cross-platform. It works on Windows, macOS, and Linux. The SDK and runtime are OS-independent. You can build and run apps using dotnet CLI on any platform.

```
dotnet build && dotnet run
```

2. Explain the request processing pipeline in ASP.NET Core MVC.

Requests pass through middleware defined in Startup.Configure. Each middleware can handle or pass request forward. Order matters for correct behavior. It ends with routing and controller invocation.

```
app.UseRouting(); app.UseEndpoints(...);
```

3. How do you implement custom middleware?

Create a class with Invoke or InvokeAsync method. Register using app.UseMiddleware<T>(). Use RequestDelegate to call next middleware.

```
public class MyMiddleware {
  private readonly RequestDelegate _next;
  public async Task InvokeAsync(HttpContext context) {
    // Custom logic
    await _next(context);
  }
}
```

4. How does routing in ASP.NET Core differ from classic ASP.NET MVC?

ASP.NET Core uses middleware-based routing. Endpoint routing decouples routing from MVC. Classic MVC uses RouteConfig. Core uses MapControllerRoute and [Route] attributes.

```
app.UseRouting(); app.UseEndpoints(e => e.MapControllers());
```

5. How do you create and use custom tag helpers?

Inherit from TagHelper and override Process. Add [HtmlTargetElement] for HTML tag association.

```
public class EmailTagHelper : TagHelper {
  public override void Process(...) {
    output.Content.SetContent("support@example.com");
  }
}
```

6. What is the role of IApplicationBuilder and IWebHostEnvironment?

IApplicationBuilder builds the middleware pipeline. IWebHostEnvironment provides environment info and content root. Inject them in Startup to configure app behavior.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env) { }
```

7. How do you secure ASP.NET Core MVC applications?

Use HTTPS, data validation, authentication/authorization. Apply [Authorize], CSRF protection, and input sanitization. Store secrets securely using Secret Manager or Key Vault.

```
services.AddAuthentication(...); app.UseAuthentication();
```

8. What are authorization policies in ASP.NET Core?

Policies define complex authorization logic. Use services.AddAuthorization() to add policies. Apply with [Authorize(Policy = "AdminPolicy")].

options.AddPolicy("AdminPolicy", policy => policy.RequireRole("Admin"));

9. How do you implement role-based authorization?

Use roles in identity and check with [Authorize(Roles = "Admin")]. Roles can be assigned in database or code.

[Authorize(Roles = "Manager,Admin")]

10. What is policy-based authorization?

Enables more flexible logic than role-based. Policies use handlers and requirements.

options.AddPolicy("MinimumAge", policy => policy.Requirements.Add(new MinimumAgeRequirement(18)));

11. What are endpoint routing and how does it differ from conventional routing?

Endpoint routing centralizes routing and supports APIs and Razor pages. Defined in UseEndpoints() vs UseMvc() in older versions.

app.UseRouting(); app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

12. How do you implement localization and globalization?

Register IStringLocalizer and use .resx files. Middleware like UseRequestLocalization() is used.

services.AddLocalization();

13. What is Razor Pages, and how does it differ from MVC?

Razor Pages is page-focused, not controller-based. Each .cshtml file has a backing model. Ideal for simple CRUD pages.

public class IndexModel : PageModel { public void OnGet() { } }

14. How do you handle exceptions globally?

Use UseExceptionHandler or custom middleware. Log errors and show friendly pages.

app.UseExceptionHandler("/Home/Error");

15. How do you log errors in ASP.NET Core MVC?

Use built-in ILogger<T>. Inject into controllers or services. Logs go to console, files, or third-party sinks.

_logger.LogError("Something went wrong");

16. How do you use custom filters?

Inherit from ActionFilterAttribute. Override OnActionExecuting. Register globally or via attributes.

```
public class LogAction : ActionFilterAttribute {
  public override void OnActionExecuting(ActionExecutingContext context) { }
}
```

17. How can you inject services into views?

Use @inject directive in Razor views. The injected service can be used like a variable.

```
@inject IMyService myService
<p>@myService.GetValue()</p>
```

18. What is the difference between ViewComponent and PartialView?

ViewComponents are reusable units with logic and view. PartialViews are only views, no logic. ViewComponents return IViewComponentResult.

public IViewComponentResult Invoke() => View("ViewName");

19. How do you implement caching in ASP.NET Core MVC?

Use IMemoryCache or response caching middleware. Add [ResponseCache] for controller caching.

services.AddMemoryCache();

[ResponseCache(Duration = 60)]

20. What are data annotations and how are they used?
Used for model validation. Add attributes like [Required], [Range] etc.
[Required] public string Name { get; set; }
21. How do you test an ASP.NET Core MVC application?
Use xUnit + Moq for unit testing controllers/services. Use TestServer for integration testing.
var result = controller.Index(); Assert.IsType<ViewResult>(result);
22. How do you use the Repository pattern in ASP.NET Core MVC?
Create repository interface and class. Inject into controller.
public interface IProductRepo { Product Get(int id); }
public class ProductRepo : IProductRepo { ... }
23. How do you access configuration settings in a strongly typed way?
Bind config section to POCO class in Startup.cs.
services.Configure<AppSettings>(Configuration.GetSection("AppSettings"));
24. What is model binding security?
Avoid over-posting using [Bind] or view models. Prevent untrusted data from binding to sensitive fields.
public IActionResult Post([Bind("Id,Name")] Product p)
25. How do you use TempData across redirects?
TempData persists data between requests via cookies. Useful after POST-Redirect-GET pattern.
TempData["Message"] = "Saved!"; return RedirectToAction("Success");
26. What is IHttpContextAccessor and how is it used?
Used to access HttpContext in non-controller classes. Add as service and inject.
var user = _httpContextAccessor.HttpContext.User;
27. What is the difference between synchronous and asynchronous controller actions?
Async actions use Task<IActionResult>. Better for I/O-bound tasks. Improves scalability.
public async Task<IActionResult> GetDataAsync() { await db.GetAsync(); }
28. How do you implement custom validation attributes?
Inherit from ValidationAttribute. Override IsValid.

```
public class NoAdmin : ValidationAttribute {
  public override bool IsValid(object value) => value?.ToString() != "admin";
}
```

29. How do you perform dependency injection in controllers, filters, and services?
Constructor injection is used. Register in Startup.cs. Filters can use TypeFilter or IFilterFactory.
public HomeController(IMyService service) { _service = service; }
30. How do you integrate third-party libraries like AutoMapper or Serilog?
Install via NuGet. Register in Startup.cs.
services.AddAutoMapper(typeof(Startup));
Log.Logger = new LoggerConfiguration().WriteTo.File("log.txt").CreateLogger();

6–12 Years Experience (Advanced)
1. How do you implement a layered architecture in ASP.NET Core MVC?
Use separate projects for UI (MVC), Business Logic, and Data Access. Interfaces decouple each layer. Use DI to inject dependencies across layers.
services.AddScoped<IProductService, ProductService>();
services.AddScoped<IProductRepository, ProductRepository>();

2. What are the best practices for securing ASP.NET Core applications?

Use HTTPS, authentication, and authorization. Validate inputs, prevent XSS/CSRF, and use secure headers. Manage secrets with environment-specific providers.

```
app.UseHttpsRedirection();
services.AddAuthentication(...);
app.UseAuthorization();
```

3. Explain the custom model binding mechanism.

Create a class that implements IModelBinder. Use ModelBinderAttribute on the action parameter. Custom logic handles binding values from the request.

```
public class CustomBinder : IModelBinder {
  public Task BindModelAsync(ModelBindingContext context) { ... }
}
```

4. How do you write and apply a custom middleware?

Create a class with InvokeAsync(HttpContext context) and register it in the pipeline. It can read/write to request and response.

```
public class LoggingMiddleware {
  public async Task InvokeAsync(HttpContext context, RequestDelegate next) {
    Console.WriteLine(context.Request.Path);
    await next(context);
  }
}
```

5. How do you manage large-scale configuration settings in ASP.NET Core?

Use appsettings.{Environment}.json, environment variables, and Azure Key Vault. Bind config to POCOs using the Options pattern.

```
services.Configure<AppSettings>(Configuration.GetSection("AppSettings"));
```

6. How do you optimize performance in ASP.NET Core MVC apps?

Use response caching, compression, async methods, and reduce view size. Use profiling tools to detect bottlenecks.

```
services.AddResponseCaching();
app.UseResponseCaching();
```

7. Explain the Dependency Injection lifecycle in detail.

Three lifetimes: Singleton (one instance), Scoped (one per request), and Transient (new every time). Choose based on data sharing needs.

```
services.AddSingleton<ILog, Log>();
services.AddScoped<IRepo, Repo>();
services.AddTransient<IService, Service>();
```

8. How do you implement feature-based folder structure?

Group controllers, views, and models by feature. Use custom ViewLocationExpander to locate views.

```
Features/
  Orders/
    OrdersController.cs
    Views/
      Index.cshtml
```

9. How do you manage secrets in production environments?
Use Secret Manager in development and Azure Key Vault in production. Avoid hardcoding credentials in appsettings.json.
dotnet user-secrets set "ApiKey" "secret-value"

10. Explain the internals of the ASP.NET Core request lifecycle.
Starts at Kestrel, passes through middleware, routing, controller, action, result execution, and finally response writing.
Request → Middleware → Routing → Controller → Action → Response

11. How do you implement OAuth2 or OpenID Connect in ASP.NET Core?
Use IdentityServer4 or external providers. Configure AddAuthentication and AddOpenIdConnect.
```
services.AddAuthentication("Cookies")
  .AddOpenIdConnect("oidc", options => { ... });
```

12. What are the different ways to handle authorization in a microservices architecture?
Use JWTs with claims, API Gateways, and policy-based authorization in services. Shared identity provider issues tokens.
```
[Authorize(Policy = "CanAccessOrders")]
```

13. How do you handle versioning in RESTful APIs with ASP.NET Core?
Use Microsoft.AspNetCore.Mvc.Versioning package. Support version in header, URL, or query string.
```
services.AddApiVersioning(options => { options.AssumeDefaultVersionWhenUnspecified = true; });
```

14. How do you use the options pattern in ASP.NET Core?
Create a POCO class, bind it to a config section, and inject using IOptions<T>.
```
services.Configure<MyOptions>(Configuration.GetSection("MyOptions"));
```

15. How do you set up and use Health Checks?
Use AddHealthChecks() and expose an endpoint. Use UI or Prometheus for monitoring.
```
services.AddHealthChecks();
app.UseEndpoints(endpoints => { endpoints.MapHealthChecks("/health"); });
```

16. How do you implement action result caching and cache profiles?
Use [ResponseCache] attribute with CacheProfile. Configure profiles in Startup.cs.
```
services.AddControllersWithViews(options =>
  options.CacheProfiles.Add("Default", new CacheProfile { Duration = 60 }));
```

17. How do you integrate Swagger in ASP.NET Core MVC?
Use Swashbuckle.AspNetCore. Register Swagger in Startup.cs and expose endpoints.
```
services.AddSwaggerGen();
app.UseSwagger(); app.UseSwaggerUI(c => c.SwaggerEndpoint(...));
```

18. How do you create unit and integration tests for MVC controllers?
Use xUnit and Moq for unit tests. Use WebApplicationFactory<T> for integration tests.
```
var result = await controller.Index();
Assert.IsType<ViewResult>(result);
```

19. What are the different hosting models in ASP.NET Core?

Self-hosted using Kestrel, or reverse proxy with IIS/Nginx. .NET Core is cross-platform and can run standalone.

dotnet run --urls "http://localhost:5001"

20. How does the Kestrel web server work in ASP.NET Core?

Kestrel is the default cross-platform server. It's a lightweight, fast, and non-blocking HTTP server based on libuv or sockets.

"urls": "https://localhost:5001"

21. How do you configure HTTPS and HSTS in ASP.NET Core?

Use UseHttpsRedirection and UseHsts in production. Enforce in middleware.

```
app.UseHsts();
app.UseHttpsRedirection();
```

22. How do you enforce CORS policies?

Use AddCors and UseCors. Define policies in Startup.cs for specific domains.

```
services.AddCors(options => {
  options.AddPolicy("MyPolicy", builder => builder.WithOrigins("https://client.com"));
});
```

23. What are anti-patterns in ASP.NET Core development?

Examples: service locator usage, fat controllers, tightly coupled layers, missing async/await, shared DbContexts.

Avoid: var service = HttpContext.RequestServices.GetService(typeof(...));

24. How do you implement tenant-based architecture (multi-tenancy)?

Use tenant resolution middleware and store tenant-specific settings in database or config.

```
public class TenantMiddleware {
  public async Task Invoke(HttpContext context) {
    var tenantId = context.Request.Headers["Tenant-Id"];
    // Set tenant context
    await _next(context);
  }
}
```

25. What is the IHostedService interface and its use case?

Used for background services. Implement StartAsync and StopAsync. Use BackgroundService for long-running tasks.

```
public class Worker : BackgroundService {
  protected override Task ExecuteAsync(CancellationToken token) { ... }
}
```

26. How do you configure multiple environments (dev, staging, prod)?

Use launchSettings.json, appsettings.{env}.json, and ASPNETCORE_ENVIRONMENT variable.

```
set ASPNETCORE_ENVIRONMENT=Production
```

27. How do you implement a custom logger provider?

Implement ILoggerProvider and ILogger, register in Startup.cs.

```
public class MyLoggerProvider : ILoggerProvider {
  public ILogger CreateLogger(string categoryName) => new MyLogger();
}
```

28. How do you profile and monitor ASP.NET Core applications?

Use tools like Application Insights, ELK, Prometheus. Use ILogger, counters, and health checks.

```
_logger.LogInformation("Start request for /api/data");
```

29. How do you implement domain-driven design (DDD) with ASP.NET Core?

Create domain models, value objects, aggregates, and services. Use repository and unit of work patterns.

```
Domain/
  Order.cs
  IOrderRepository.cs
Application/
  PlaceOrderHandler.cs
```

30. How do you use SignalR in conjunction with MVC?

Add SignalR services and hub. Use JS client in Razor Views.

```
services.AddSignalR();
app.UseEndpoints(endpoints => { endpoints.MapHub<ChatHub>("/chatHub"); });
```

12–18 Years Experience (Expert/Architect Level)

1. How do you design and scale enterprise-grade ASP.NET Core MVC applications?

Use modular architecture with bounded contexts and DDD principles, separating layers (UI, application, domain, infrastructure). Use microservices or horizontal scaling behind load balancers. Utilize caching, database sharding/replication, and container orchestration. Use CI/CD pipelines and monitoring for full-stack performance and reliability.

```
services.AddSwaggerGen(); services.AddHealthChecks();
```

2. How do you architect microservices with ASP.NET Core MVC and APIs?

Define each service with single responsibility, independently deployable. Use lightweight communication (HTTP/gRPC), centralized configuration, and distributed tracing. Implement API gateways for routing and security.

```
// appsettings.json
"ServiceDiscovery": { "Consul": "http://consul:8500" }
```

3. Explain hybrid monolith + microservices strategy in ASP.NET Core.

Maintain a monolithic core for shared concerns while extracting high-value areas as microservices. Use shared libraries and versioned APIs. Gradually migrate to distributed architecture.

```
app.MapControllerRoute("users", "{controller=User}/{action=Index}/{id?}");
```

4. How do you apply SOLID principles and design patterns in ASP.NET Core MVC?

Use Dependency Inversion for loose coupling, Single Responsibility Controllers, Strategy and Repository patterns for business logic and data access. Use Factory or Decorator for extensibility.

```
public IOrderService OrderService { get; } // injected via constructor
```

5. How do you implement CQRS and Mediator patterns?

Separate read/write workloads: commands via Mediator (IMediator.Send(cmd)), queries via Mediator (IMediator.Query(qry)). Use libraries like MediatR for orchestration and pipeline behaviors.
await mediator.Send(new CreateOrderCommand { ... });

6. How do you integrate ASP.NET Core MVC with distributed systems?

Use message brokers like RabbitMQ or Kafka for async integration, sidecars for reliability. Use health checks, service mesh (e.g., Istio), and resiliency patterns (retries, fallbacks, circuit breakers).
services.AddHostedService<QueueListenerService>();

7. How do you manage identity and access in multi-service ASP.NET Core apps?

Use centralized IdentityServer or Azure AD for issuing JWTs or reference tokens. Each service validates tokens and enforces scopes/roles. Use token caching and refresh mechanisms.
services.AddAuthentication("Bearer").AddJwtBearer(...);

8. What is the role of gRPC vs REST in modern ASP.NET Core solutions?

REST over HTTP/JSON is universal and easy for public APIs; gRPC is high-performance, strongly typed, and ideal for inter-service communication. Mixing both depending on performance/security requirements.
service ProductService { rpc GetProduct (GetProductRequest) returns (ProductDto); }

9. How do you ensure observability (logs, metrics, traces) in ASP.NET Core?

Integrate OpenTelemetry, ILogger, Prometheus metrics, structured logging (Serilog), and distributed tracing (Jaeger). Emit contextual logs with correlation IDs across services.
builder.Logging.AddOpenTelemetry(o => o.AddConsole());

10. What are strategies for zero-downtime deployment in ASP.NET Core?

Use blue-green or canary deployment via load balancers or Kubernetes. Ensure backward compatibility at API and DB migration levels. Use health probes to route traffic only to healthy instances.
readinessProbe:
httpGet: path: /healthz

11. How do you use Kubernetes and Docker with ASP.NET Core MVC?

Dockerize app using multi-stage Dockerfile, push to registry. Deploy via Helm charts with config maps, secrets, HPA, and readiness/liveness probes.
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime

12. How do you implement advanced validation pipelines using FluentValidation?

Use FluentValidation validators bound to request models. Integrate via AddFluentValidation() and customize with rulesets.
public class OrderValidator : AbstractValidator<OrderDto> {
RuleFor(x => x.Quantity).GreaterThan(0);
}

13. How do you handle message queues (RabbitMQ, Azure Service Bus) in ASP.NET Core?

Inject queue clients, implement background services for listeners, use retry policies. Acknowledge or dead-letter messages appropriately.
services.AddSingleton<IMessagePublisher, RabbitMqPublisher>();

14. How do you enforce Domain-Driven Design boundaries in large applications?

Use distinct projects and namespaces per domain, enforce encapsulation in aggregates, avoid sharing data objects across boundaries, and use domain events for communication.
public class Order : Entity, IAggregateRoot { ... }

15. How do you build and expose APIs for external systems securely?

Use OAuth2/JWT authentication, rate limiting, input validation, TLS/SSL, and API gateways for WAF protection. Document via Swagger with secure endpoints.
builder.Services.AddRateLimiter(options => options.AddFixedWindowLimiter(...));

16. How do you implement advanced caching with Redis or distributed memory cache?

Use StackExchangeRedis, IDistributedCache. Use sliding expiration and cache invalidation logic.

```
await cache.SetStringAsync("key", value, new DistributedCacheEntryOptions{
AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5)});
```

17. How do you use circuit breakers and resiliency policies with Polly?

Wrap HTTP client or DB calls with AddPolicyHandler() for retry, fallback, and circuit-breaker policies.

```
AddHttpClient("api").AddTransientHttpErrorPolicy(p => p.CircuitBreakerAsync(3,
TimeSpan.FromMinutes(1)));
```

18. How do you deal with GDPR and data protection in ASP.NET Core apps?

Use DataProtection for encryption at rest, consent screens for cookies, data deletion and retention policies, record data access via audit trails.

```
services.AddDataProtection().PersistKeysToAzureBlobStorage(...);
```

19. How do you enforce secure coding practices across large teams?

Use code analysis tools (SonarQube, Roslyn), enforce PR reviews, automated security testing (SAST), coding standards, and continuous security training.

```
- task: SonarQubeAnalyze@5
```

20. How do you choose between Razor Pages, MVC, and Blazor for large systems?

Use Razor Pages for page-focused scenarios, MVC for API-based web apps, and Blazor for interactive SPAs. Consider team expertise, interactivity, and maintainability. No code example.

21. What is your strategy for code reviews and CI/CD in large ASP.NET Core projects?

Use Git branching strategies (GitFlow, Trunk-based), automated builds and tests in pipelines, and gated PRs. Use environment-specific deployment stages with proper rollback handling.

```
trigger:
- main
```

22. How do you structure solutions for maintainability and modularity?

Use folder-by-feature structure, clean architecture with domain, application, infrastructure, web projects. Use NuGet for shared libraries.

```
src/
  Domain/
  Application/
  Infrastructure/
  WebApi/
```

23. How do you handle breaking changes in APIs with backward compatibility?

Version APIs via URL or media type. Maintain default version and deprecate gracefully, use API gateways for transformation.

```
[ApiVersion("1.0")]
[ApiController]
```

24. What are strategies for blue-green or canary deployment in ASP.NET Core?

Deploy two identical environments (blue/green) and switch traffic. Canary uses percentage-based gradual rollout via service mesh or API gateway. No code.

25. How do you handle cross-cutting concerns (logging, validation, auth) effectively?

Use middleware, filters, or decorator patterns. Centralize policies using DI and configure services in startup.

```
services.AddControllers(options => options.Filters.Add<ValidationFilter>());
```

26. How do you build a platform-as-a-service architecture using ASP.NET Core?

Offer shared components (auth, logging, messaging) as reusable services. Use customizable templates and tenant isolation using subdomains or paths. No code.

27. How do you plan and execute performance tuning at scale?

Use load testing (JMeter), profiling (dotTrace), caching, clustering, DB indexing, and analyzing telemetry to identify hot spots. Continuously benchmark improvements.

28. What role does gRPC-Web or GraphQL play in modern ASP.NET Core systems? gRPC-Web enables browser-compatible gRPC interactions. GraphQL provides flexible querying schemas, reduces over-fetching, useful for complex client scenarios. No code.

29. How do you manage versioning and documentation in large API ecosystems? Use OpenAPI specs, automated Swagger generation, versioned API endpoints, continuous documentation pipelines and publish with semantic versioning.

services.AddSwaggerGen(c => c.SwaggerDoc("v1", new OpenApiInfo {Version="v1"}));

30. How do you mentor and architect solutions across globally distributed ASP.NET Core teams? Enforce architecture guidelines via reference implementations, hold regular design reviews, pair programming across regions, and maintain shared engineering culture with documentation and knowledge bases. No code.