# Angular Ivy

Ivy is the code name for Angular's next-generation compilation and rendering pipeline. With the version 9 release of Angular, the new compiler and runtime instructions are used by default instead of the older compiler and runtime, known as View Engine.

> Learn more about the Compiler and Runtime in these videos from our team.

## AOT and Ivy

AOT compilation with Ivy is faster and should be used by default. In the `angular.json` workspace configuration file, set the default build options for your project to always use AOT compilation. When using application internationalization (i18n) with Ivy, translation merging also requires the use of AOT compilation.

angular.json

```
{
  "projects": {
    "my-existing-project": {
      "architect": {
        "build": {
          "options": {
            ...
            "aot": true,
          }
        }
      }
    }
  }
}
```

## Ivy and libraries

Ivy applications can be built with libraries that were created with the View Engine compiler. This compatibility is provided by a tool known as the Angular compatibility compiler (`ngcc`). CLI commands run `ngcc` as needed when performing an Angular build.

For more information on how to publish libraries see Publishing your Library.

### Maintaining library compatibility

If you are a library author, you should keep using the View Engine compiler as of version 9. By having all libraries continue to use View Engine, you will maintain compatibility with default v9 applications that use Ivy, as well as with applications that have opted to continue using View Engine.

See the Creating Libraries guide for more on how to compile or bundle your Angular library. When you use the tools integrated into the Angular CLI or `ng-packagr`, your library will always be built the right way automatically.

## Ivy and Universal/App shell

In version 9, the server builder which is used for App shell and Angular Universal has the `bundleDependencies` option enabled by default. If you opt-out of bundling dependencies you will need to run the standalone Angular compatibility compiler (`ngcc`). This is needed because otherwise Node will be unable to resolve the Ivy version of the packages.

You can run `ngcc` after each installation of node_modules by adding a `postinstall` npm script:

package.json

```
{
  "scripts": {
    "postinstall": "ngcc"
  }
}
```

> - The `postinstall` script will run on every installation of `node_modules`, including those performed by `ng update` and `ng add`.
> - Don't use `--create-ivy-entry-points` as this will cause Node not to resolve the Ivy version of the packages correctly.

## Opting out of Ivy in version 9

In version 9, Ivy is the default. For compatibility with current workflows during the update process, you can choose to opt out of Ivy and continue using the previous compiler, View Engine.

> Before disabling Ivy, check out the debugging recommendations in the Ivy Compatibility Guide.

To opt out of Ivy, change the `angularCompilerOptions` in your project's TypeScript configuration, most commonly located at `tsconfig.app.json` at the root of the workspace.

The value of the `enableIvy` flag is set to `true` by default, as of version 9.

The following example shows how to set the `enableIvy` option to `false` in order to opt out of Ivy.

tsconfig.app.json

```
{
  "extends": "./tsconfig.json",
  "compilerOptions": {
    "outDir": "./out-tsc/app",
    "types": []
  },
  "files": [
    "src/main.ts",
    "src/polyfills.ts"
  ],
  "include": [
    "src/**/*.d.ts"
  ],
  "angularCompilerOptions": {
    "enableIvy": false
  }
}
```

> If you disable Ivy, you might also want to reconsider whether to make AOT compilation the default for your application development, as described above.
>
> To revert the compiler default, set the build option `aot: false` in the `angular.json` configuration file.

If you disable Ivy and the project uses internationalization, you can also remove the `@angular/localize` runtime component from the project's polyfills file located be default at `src/polyfills.ts`.

To remove, delete the `import '@angular/localize/init';` line from the polyfills file.

polyfills.ts

```
/***************************************************************************************************
 * Load `$localize` onto the global scope - used if i18n tags appear in Angular templates.
 */
import '@angular/localize/init';
```

### Using SSR without Ivy

If you opt out of Ivy and your application uses Angular Universal to render Angular applications on the server, you must also change the way the server performs bootstrapping.

The following example shows how you modify the `server.ts` file to provide the `AppServerModuleNgFactory` as the bootstrap module.

- Import `AppServerModuleNgFactory` from the `app.server.module.ngfactory` virtual file.
- Set `bootstrap: AppServerModuleNgFactory` in the `ngExpressEngine` call.

server.ts

```
import 'zone.js/dist/zone-node';

import { ngExpressEngine } from '@nguniversal/express-engine';
import * as express from 'express';
import { join } from 'path';

import { APP_BASE_HREF } from '@angular/common';

import { AppServerModuleNgFactory } from './src/app/app.server.module.ngfactory';

// The Express app is exported so that it can be used by serverless Functions.
export function app() {
  const server = express();
  const distFolder = join(process.cwd(), 'dist/ivy-test/browser');

  // Our Universal express-engine (found @ https://github.com/angular/universal/tree/master/modules/express-engine)
  server.engine('html', ngExpressEngine({
    bootstrap: AppServerModuleNgFactory,
  }));

  server.set('view engine', 'html');
  server.set('views', distFolder);

  // Example Express Rest API endpoints
  // app.get('/api/**', (req, res) => { });
  // Serve static files from /browser
  server.get('*.*', express.static(distFolder, {
    maxAge: '1y'
  }));

  // All regular routes use the Universal engine
  server.get('*', (req, res) => {
    res.render('index', { req, providers: [{ provide: APP_BASE_HREF, useValue: req.baseUrl }] });
  });

  return server;
}

function run() {
  const port = process.env.PORT || 4000;

  // Start up the Node server
  const server = app();
  server.listen(port, () => {
    console.log(`Node Express server listening on http://localhost:${port}`);
  });
}

// Webpack will replace 'require' with '__webpack_require__'
// '__non_webpack_require__' is a proxy to Node 'require'
// The below code is to ensure that the server is run only when not requiring the bundle.
declare const __non_webpack_require__: NodeRequire;
const mainModule = __non_webpack_require__.main;
if (mainModule && mainModule.filename === __filename) {
  run();
}

export * from './src/main.server';
```